ESCUELA SUPERIOR POLITECNICA DEL LITORAL

FACULTAD DE INGENIERIA EN ELECTRICIDAD Y COMPUTACION

"DESARROLLO DE APLICACION TELEFONICA UTILIZANDO LAS LIBRERIAS TELEFONICAS DE WINDOWS TAPI CON SOPORTE DE CALLER ID PARA LAS CENTRALES QUE UTILIZAN EL STANDART BELLCORE EN ECUADOR"

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN ELECTRICIDAD

ESPECIALIZACION:

ELECTRONICA

PRESENTADO POR:

Edmundo Ricardo Vera Bonilla

GUAYAQUIL - ECUADOR

2001

AGRADECIMIENTO

A la Ing. Rebeca Estrada, Profesora Supervisora de este proyecto, por su ayuda, colaboración y paciencia para el desarrollo de este trabajo

DEDICATORIA

A mis amigos de toda la vida.

A mis padres y hermanas.

A mi hijo y esposa.

TRIBUNAL DE GRADUACIÓN

ING. CARLOS MONSALVE

SUBDECANO

FACULTAD DE INGENIERIA ELECTRICA

ING. REBECA ESTRADA

DIRECTOR DE TESIS

ING. FREDDY VILLAO QUEZADA

MIEMBRO DE TRIBUNAL

ING. PEDRO VARGAS

MIEMBRO DE TRIBUNAL

Declaración Expresa

"La responsabilidad por los hechos , ideas y doctrinas expuestos en este informe técnico me corresponden exclusivamente, y el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

"Reglamento de Exámenes y Títulos profesionales de la ESPOL"

Edmundo Ricardo Vera Bonilla

RESUMEN

Las capacidades del sistema de Aplicaciones de Telefonía de la compañía Microsoft, TAPI, como modelar las líneas físicas de telefonía a usarse para enviar y recibir voz y datos entre localizaciones, manejar el medio físico usado (el teléfono de mesa por ejemplo) para colocar y recibir llamadas. Por esta razón, he considerado la implementación de un programa de manejo telefónico para uso doméstico u oficinas en empresas pequeñas y medianas, que puede suplir el dispositivo telefónico, en donde su uso sea frecuente y existan las condiciones de optimizarlo por medio de un computador. Además presentaré un análisis las capacidades de un usuario final para desarrollar programas sencillos, pero de gran potencial y alcance importantes.

El modelo de API de telefonía, TAPI, es un simple conjunto de funciones predeterminado que pueden usarse para ganar acceso a todos los aspectos de los servicios de telefonía dentro del sistema operativo de Windows. La meta de TAPI es permitir a los programadores escribir aplicaciones que trabajen a pesar del medio físico telefónico disponible para la PC. Las aplicaciones escritas pueden trabajar de forma idéntica tanto en líneas analógicas o digitales y telefónicas, además de que pueden comunicarse con sofisticados terminales digitales multilínea usados en oficinas de altas tecnologías.

La aplicación desarrollada es una solución completa de dispositivo telefónico. Con un módem que soporte datos de voz, una tarjeta de sonido adecuada y este programa funcionando en su máquina, se puede eliminar completamente el dispositivo telefónico de su escritorio. Como parte del

programa, se puede manejar llamadas entrantes y salientes. Se dispone de contestador automático, grabadora de mensajes y código remoto de acceso a los mensajes almacenados desde cualquier otra localidad. También se ha dado acceso a los usuarios a diversas ventanas de diálogo TAPI para mantener una configuración para la aplicación, incluyendo mantenimiento de una pequeña base de datos telefónica tanto de mensajes grabados, llamadas entrantes, historial de llamadas con rumbo exterior y libreta telefónica personal. Mediante TAPI, se ha logrado que el soporte para Caller Id sea completo, especialmente para manejar el estándar Bellcore, el cual es el utilizado por las centrales telefónicas en Ecuador.

Índice

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El presente proyecto tiene como objetivo principal de introducir al usuario doméstico y técnico al sistema de Aplicaciones de Telefonía desarrollado por la compañía Microsoft, TAPI, debido a la importancia que representa los sistemas de telecomunicaciones en el mundo. En un mundo competitivo, cada vez es mayor la necesidad de eficiencia tecnológica y económica. Con este proyecto se pretende mostrar formas básicas para poder desarrollar solución es para las nuevas capacidades de telecomunicaciones que se están desarrollando actualmente. Se analizara principalmente las capacidades de un usuario final para desarrollar sencillos programas, pero de potencial y alcance importantes. Tomaremos como servicio principal de ejemplo al servicio de identificador de llamadas Caller Id. Finalmente, desarrollaremos un software de aplicación telefónica para un usuario del tipo doméstico o de una pequeña empresa, de tal manera que pueda realizar un amplio manejo telefónico para su uso doméstico o en oficinas de empresas pequeñas y medianas, en donde el uso del dispositivo telefónico sea frecuente y en condiciones de ser optimizado por medio de un computador.

1.1. Antecedentes

Las telecomunicaciones son parte vital de los países industrializados, y objetivo primordial de desarrollo en los países en búsqueda de avances. Gran parte del importante desarrollo actual del Ecuador se basa en la modernización de sus sistemas de telecomunicaciones. Así, se busca desarrollar una mejor infraestructura y proveer una amplia gama de nuevos servicios, desde los complejos y amplios para industrias, hasta los pequeños servicios del tipo usuario final, cuyo objetivo final será siempre mejorar el rendimiento del uso de las telecomunicaciones, y así aumentar la productividad general del área industrial.

Sin embargo, parte del proceso de aumento de competitividad y eficiencia industrial debe ser el entrenamiento y la proporción de recursos para desarrollar soluciones personalizadas para utilizar las nuevas capacidades de telecomunicación existentes.

1.2. Objetivos generales

El objetivo del presente proyecto es el de instruir sobre las capacidades del sistema de Aplicaciones de Telefonía (TAPI), para de esta manera poder desarrollar soluciones para las nuevas capacidades de telecomunicaciones que se necesitan actualmente. Se analizará principalmente las capacidades de un usuario final para desarrollar sencillos programas, pero de gran potencial y alcance importantes. Se aplicará estos conceptos a un programa de manejo telefónico para uso doméstico u oficinas en empresas pequeñas y medianas, en donde el uso del dispositivo telefónico sea frecuente y en condiciones de ser optimizado por medio de un computador. Este programa tendrá soporte para uno de los principales y

más nuevos servicios implementados, como es el servicio de identificador de llamadas (Caller Id).

1.3. Objetivos Específicos

- Dar una visión general del modelo de Arquitectura Abierta de Servicios Windows (WOSA).
- Introducir el concepto de TAPI y su programación.
- Presentar el funcionamiento general del servicio de Caller Id.
- Desarrollar un programa que maneje el modem del computador, y realice operaciones telefónicas cotidianas, incluyendo soporte para el servicio Caller Id y máquina contestadora automática.

1.4. Contribución

Una solución telefónica basada en las funciones TAPI, con capacidades extendidas, por ejemplo el soportar detección de Caller Id. De esta manera se busca contribuir con el área tecnológica al difundir un sistema práctico de desarrollo de solución en telecomunicaciones y proporcionar un software de aplicación real para un amplio número de usuarios.

Por otro lado, se motiva la instrucción de las personas del medio para aumentar su rendimiento y capacidades de desarrollo autónomo. El programa es básicamente para un usuario final doméstico, aunque por su funcionalidad como máquina contestadora automática, grabadora de mensajes y su amplio manejo de bases de datos, puede ser de ayuda en un negocio de dimensiones pequeñas y medianas.

1.5. Perfil de la tesis

En el segundo capítulo iniciaremos el estudio de TAPI, mediante una introducción al sistema WOSA y sus principios de funcionamiento, que es de donde se desprende el servicio TAPI. Después proseguiremos analizando específicamente la arquitectura del modelo TAPI, sus tipos de servicios y funciones, y la forma en que debe ser implementado dentro de una aplicación telefónica especifica.

El capítulo tres presenta el conjunto de protocolos del servicio Caller Id. Básicamente estudiaremos los tipos de protocolos y convecciones para este servicio, para proseguir con un análisis de su trama de datos y su funcionamiento en diferentes medios telefónicos.

El capitulo cuatro será dedicado a presentar los servicios extendidos de Tapi. Básicamente estos son servicios especiales desarrollados para otros varios tipos de dispositivos. Estas funciones serán presentadas y explicadas para exponer al modelo TAPI en su totalidad, pues muestra su valor como dispositivo de desarrollo extendido de aplicaciones.

En el capítulo cinco se explicará en detalle el desarrollo del software telefónico, para el cual se ha utilizado las librerías TAPI para implementar una aplicación servicios telefónicos extendidos para una estación o PC de escritorio, conectada a una línea telefónica por medio de un módem con soporte de sonido. Además, como servicio agregado a esta aplicación telefónica de escritorio, se ha implementado soporte para Caller Id.

Finalmente, el capítulo seis expone las conclusiones y las recomendaciones ideadas para el seguimiento de este proyecto en un futuro cercano.

Capítulo 2

TAPI

El modelo de la Arquitectura Abierta de Servicios Windows (WOSA) fue desarrollado por la corporación Microsoft con el concepto de diseñar una forma de utilizar servicios extendidos del sistema operativo Windows, de una forma tal que solo requiera una mínima cantidad de información acerca de cómo funcionan estos servicios realmente.

El modelo de API de telefonía TAPI es diseñado para proveer acceso a servicios telefónicos en todas las plataformas de Windows. La API de telefonía es un simple conjunto de funciones, cuya meta es permitir a los programadores escribir aplicaciones que trabajen a pesar del medio físico telefónico disponible para la computadora personal (PC). Las aplicaciones escritas con TAPI pueden trabajar de forma idéntica tanto en líneas analógicas, digitales y telefónicas. En este capítulo estudiaremos únicamente las funciones básicas de TAPI.

2.1. Introducción de Arquitectura Abierta de Servicios Windows (WOSA)

El concepto de WOSA es diseñar una forma de acceder a servicios extendidos del sistema operativo Windows, de una forma tal que solo requiera una mínima cantidad de información acerca de cómo los servicios trabajan realmente. Por ejemplo, el API de Mensajería (MAPI) esta diseñado para permitir a programadores desarrollar aplicaciones que usan los servicios de mensajería, sin tener que entender las complejidades del hardware y las rutinas de software que implementan el envío de mensajes en diversas plataformas Windows. Simplemente, siguiendo un procedimiento de comunicación y un sistema determinado de pasos para comunicarse con el modelo WOSA, se instruye al sistema operativo para que realice las actividades. Finalmente, el que realiza el trabajo de lidiar con el hardware y realizar el control del servicio es el sistema WOSA. Así, generar programas potentes y de gran alcance es sencillo, al ser necesario únicamente entender el proceso de comunicación con WOSA, dejando de esta manera el resto del trabajo y administración del servicio al sistema operativo Windows. Lo mismo es valido para cualquier otro tipo de API de WOSA, como el API de Discurso-voz (SAPI) y los servicios de Telefonía (TAPI), los cuales comprenderemos en este capítulo.

2.1. El Modelo WOSA

Para lograr esta flexibilidad, el modelo WOSA se separa en 2 interfaces o partes distintas: la aplicación del Cliente (API) y la aplicación del servidor (SPI). Estos están vinculados por un solo camino o interfaz, que se encarga de enviar la información de contacto entre las aplicaciones API y SPI. Como consecuencia, lo que toda aplicación diseñada para el tipo cliente

necesita hacer, es seguir las reglas definidas por la API dominante y proseguir para la interfaz universal. Toda aplicación de servidor, de forma similar, solo necesita contactar al SPI dominante y proseguir entonces para la interfaz universal. No importa que cambios se realicen en las aplicaciones del cliente o del servidor, ambos módulos informáticos (el cliente y el servidor) serán compatibles mientras ambos continúen utilizando el modelo API/SPI y usando la interfaz universal. Los tres componentes WOSA son en resumen:

- **El cliente API.** La interfaz de programación de la aplicación utilizada por el programa que requiere el servicio. Son las reglas que debe seguir un programa por el lado del cliente para pedir servicios.
- **El servidor SPI.** La interfaz del proveedor del servicio usada por el programa que provee el servicio extendido (por ejemplo, el correo electrónico, la telefonía, los servicios de discurso, etcétera). Son las reglas que debe seguir un programa para utilizar los recursos del servidor.
- **La interfaz API/SPI.** El modulo que conecta el API y las llamadas SPI. Esto es usualmente implementado como un DLL separado en el ambiente de Windows.

La figura 2.1 demuestra la relación entre los tres componentes WOSA.

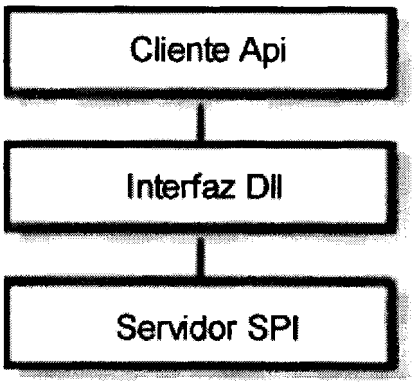

Figura 2.1 Relación entre los componentes WOSA

2.2.1 API del Cliente realiza requerimientos

El Cliente API es la interfaz para la aplicación que demanda el servicio. Son usualmente llamadas implementadas desde el escritorio de Windows, o mejor dicho, son las funciones que recogen los requerimientos del cliente para después mandarlas al servidor vía interfaz Dll. Estos servicios son solamente demandados por el cliente, pero los servicios reales serán provistos por la aplicación en servidor.

Aquí justamente radica el punto crucial. La interfaz cliente permite a los programas demandar servicios, pero no permite que el software del cliente gane acceso a los servicios subyacentes directamente, pues la petición no es mandada directamente para la aplicación en servidor. Esta es enviada a la interfaz DLL que se sienta entre la API y SPI (ver Figura No 2.2).

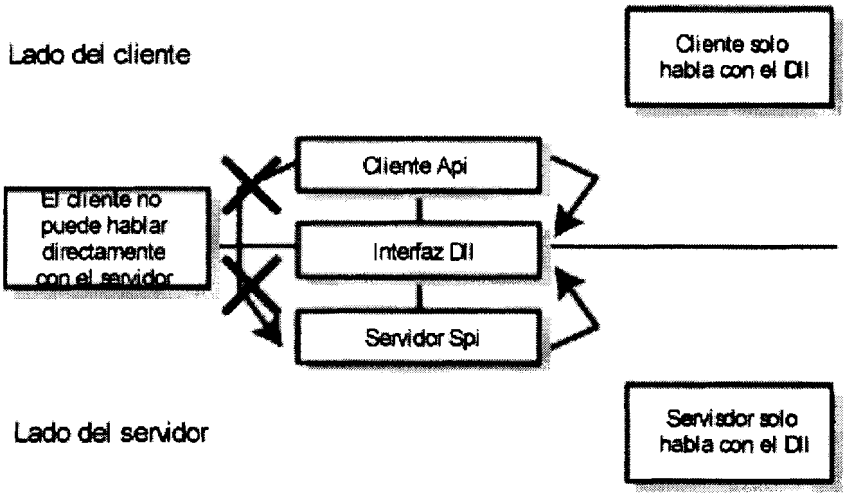

Figura 2.2 Comunicación entre elementos WOSA

Esto aísla las llamadas de API y de SPI y limita la posibilidad de que cambios futuros para uno o el otro componente afecten adversamente a la interfaz.

2.2.2 Servidor SPI Responde las Requerimientos

El Servidor SPI (Interfaz del proveedor de servicios - driver) acepta requerimientos para los servicios y actúa siguiendo esos requerimientos.

Como mencionamos anteriormente, los proveedores de servicios raramente realizan una conexión de comunicación directamente con las aplicaciones del cliente. Su trabajo es responder al requerimiento y realizar el trabajo pedido. Incluso, estos requerimientos pueden no venir directamente del programa del cliente. Este es otro punto importante que debe ser puesto de relieve. La parte "proveedor de servicios" del modelo WOSA permite tener en cuenta las demandas de servicios provenientes de clientes múltiples.

Cualquier información que el SPI necesita suministrar como una parte de la respuesta para la petición es enviada igualmente para la interfaz DLL. Es el trabajo de DLL enviar la información de respuesta para el cliente que hizo la petición inicial.

La interfaz DLL dice al SPI que cliente realizo la petición. Es responsabilidad del SPI mantener a los clientes diversos en comunicación. Los SPI's deben tener la habilidad para manejar tanto requerimientos múltiples del mismo cliente como de clientes múltiples.

2.2.3 La Interfaz DLL se comunica con ambos API y SPI

Esta interfaz es usualmente implementada como una biblioteca dinámica Windows (DLL), que permite al programa relacionarse con servicios en proceso de corrida en lugar de compilar una interfaz nueva. La ventaja de usar DLLs es que los programas no necesitan saber todo acerca de una interfaz en la fase de compilación. Así, los programadores pueden actualizar módulos DLL sin tener que recompilar las aplicaciones que ganan acceso a la interfaz.

El papel primario del DLL es conectar y manejar los requerimientos del API del cliente y las respuestas del SPI. El DLL realmente no realiza ningún servicio real para el cliente y tampoco realiza requerimientos para el SPI. Realmente, la interfaz DLL puede demandar información básica del SPI en el arranque acerca de los servicios subyacentes SPI, su disponibilidad, y otra información que puede ser necesaria para respaldar requerimientos de clientes. La interfaz DLL es la única aplicación en el ambiente de Windows que realmente "habla" con API y SPI. Es justamente el trabajo de DLL actuar como traductor entre el cliente y las aplicaciones del servidor.

2.3 Beneficios de WOSA

Hay varios beneficios para ambos usuarios y los programadores en el modelo WOSA. Los tres beneficios cruciales que vale mencionar aquí son:

- El desarrollo aislado
- El apoyo a múltiples vendedores de hardware y software
- La protección de actualización

Todos los beneficios de WOSA son resultado de la habilidad del modelo para separar los detalles de los proveedores de servicios, de la aplicación corriendo en los escritorios de los usuarios. Conservando los detalles de hardware y software cerrado en el lado del SPI, los programadores pueden concentrarse en proveer una interfaz consistente para los servicios.

Los desarrolladores pueden limitar su inversión de tiempo en entender los detalles de una tecnología de servicio. Aquellos que se enfocan en desarrollar aplicaciones en cliente, pueden dejar los detalles del desarrollo de servicios en servidor para otros. Ellos podrán concentrar sus esfuerzos en desarrollar software de calidad en cliente, pues mientras los proveedores de servicios mantengan compatibilidad con WOSA, el software del cliente podrá aprovechar servicios nuevos cuando se pongan disponibles. Esto trabaja similar para desarrolladores de software en servidor, al concentrarse únicamente en encontrar la manera más eficiente y efectiva de exponer y respaldar los requerimientos de servicios

Los usuarios pueden implementar más fácilmente mejoras del software y del hardware, porque las llamadas de acceso al servicio están

aisladas en un solo DLL. . Los desarrolladores pueden confiar que mientras los proveedores de servicios mejorares y actualicen su software, no hay necesidad de que las aplicaciones del cliente sean reescritas, excepto para agregar soporte que aproveche los servicios nuevos del software o el hardware, pues el modelo WOSA asegura que el acceso a los servicios es estándar a través de cualquier producto y plataforma.

2.4 El Modelo API de Telefonía (TAPI)

Como parte del modelo WOSA, TAPI provee una implementación completa de telefonía para sistemas operativos Windows sin forzar al programador a aprender funciones específicas de los vendedores-proveedores de los dispositivos.

El modelo del diseño TAPI está dividido en dos áreas, cada uno con su propio conjunto de funciones de llamadas de API. Cada conjunto de funciones de API enfoca su atención en lo que TAPI se refiere como a un dispositivo. Los dos dispositivos TAPI son:

- Los dispositivos de la línea: modelan las líneas físicas de telefonía a usarse para enviar y recibir voz y datos entre localizaciones.
- Los dispositivos telefónicos: modelan el medio físico (el teléfono de mesa) usado para colocar y recibir llamadas.

2.4.1 Las líneas

El dispositivo de la línea se usa para manejar la línea telefónica física. Es importante tener por entendido que, en TAPI, el dispositivo de la línea no es realmente una línea física; Es simplemente un modelo u objeto

representando una línea física. En aplicaciones TAPI, un programa puede seguir la pista a varios dispositivos de la línea, cada uno del cual esta conectado para una sola línea física. Es así que la misma aplicación TAPI podría seguir la pista a los dispositivos múltiples de línea (por ejemplo, un fax, un módem de datos y un dispositivo de voz), que numeraria más del total real de líneas físicas disponibles para la PC (por ejemplo, una sola línea telefónica). Si la PC hace el reconocimiento de un único medio para llamar por teléfono, entonces la aplicación TAPI compartirá la única línea entre lo tres dispositivos definidos para la línea. Esto se llama planificación de la memoria dinámica de la línea (ver Figura No 2.3).

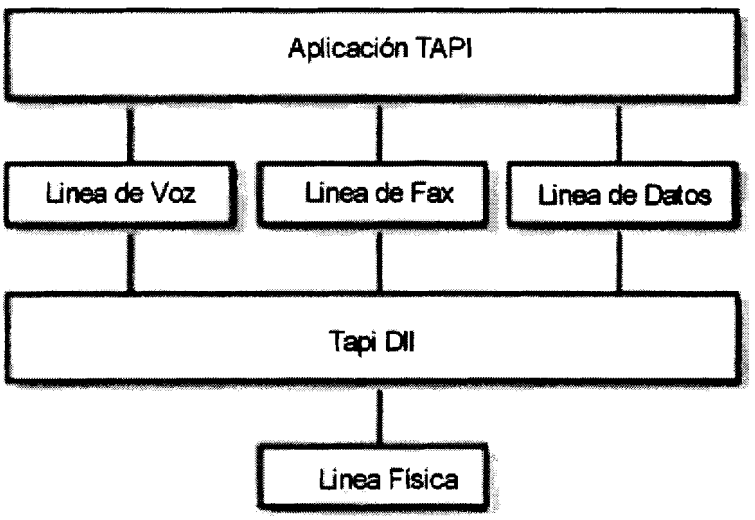

Figura 2.3 Planificación dinámica de la línea

Cada vez que la aplicación TAPI pone en marcha un dispositivo de la línea, demanda la primera línea física disponible que tiene las capacidades necesitadas para mantenerla (voz, fax, datos, etcétera). Si una línea no esta disponible, entonces un mensaje para ese efecto es devuelto para el programa del requerimiento. En algunos casos, como transmisiones por fax, la aplicación TAPI "puede hacer cola" con la petición de línea, para procesarla posteriormente.

Si dos líneas están disponibles, entonces la aplicación TAPI las usa dependiendo de como sean necesarias. Si un tercer dispositivo de línea se activa, entonces la aplicación TAPI sabrá que para aquella ya no habrá otras líneas abiertas disponibles y notifica al usuario (o puede encolar su requerimiento de llamada para más adelante).

TAPI puede también seguir la pista a que tipos de líneas están disponibles. Por ejemplo, una de las dos líneas conectadas para el PC puede ser una línea dedicada de transmisión de datos de alta velocidad (como una línea de 56Kbps), y la otra una línea del tipo "voz" (como una línea del 3.1KHz). Si la aplicación TAPI demanda unos datos de alta velocidad de línea, entonces el SPI devolviera el manejador de la línea de 56Kbps, para especificar que línea se usara. Si la aplicación demanda una línea del grado de voz, entonces el SPI devolvía el manejador de la línea de la voz. Sin embargo, si la segunda petición (línea de voz) era para también transmisión de datos, el SPI sabrá que la línea del grado de voz no es aceptable y devolverá un mensaje diciendo que todas las líneas de datos están ocupadas.

2.4.2 Los teléfonos

El segundo tipo de dispositivo modelado por TAPI es el dispositivo telefónico. Este modelo permite a los programadores TAPI fácilmente crear "teléfonos virtuales". Por ejemplo, una PC estándar con una tarjeta de sonido, parlantes, y el micrófono pueden emular todas las funciones de un teléfono de mesa. Un solo PC puede modelar varios dispositivos telefónicos, cada uno con sus únicas características. Cuando una llamada real debe ser hecha, el usuario puede seleccionar uno de los dispositivos telefónicos,

introducir el número deseado y entonces la aplicación TAPI asigna el dispositivo telefónico a un dispositivo disponible de línea. Se debe notar que los dispositivos telefónicos se complementan con los dispositivos de línea (los cuales eventualmente se conectaran con líneas telefónicas físicas).

Un ejemplo típico de la relación entre dispositivos telefónicos y dispositivos de línea es el modelo de un tablero de mando de teléfonos para la oficina. El tablero de mando típico tiene varias líneas interurbanas que terminan en un solo teléfono de la multilínea. Usualmente este teléfono tiene varias luces relampagueantes y botoneras. Los botones se usan para conectar líneas interurbanas para teléfonos de la extensión dentro de la oficina. Los teléfonos de la extensión podrían ser modelados como los dispositivos telefónicos en la pantalla de un recepcionista. Las líneas interurbanas entrantes podrían ser manejadas como dispositivos de línea. Cuando las llamadas entran, el recepcionista "la tomaría-contestaría" para luego "dejarla caer" en el dispositivo del tablero de mando, que determinaría para quién debe ser encaminada la llamada, para finalmente dirigirla fuera del tablero de mando teléfono hacia la extensión apropiada (ver Figura No 2.4).

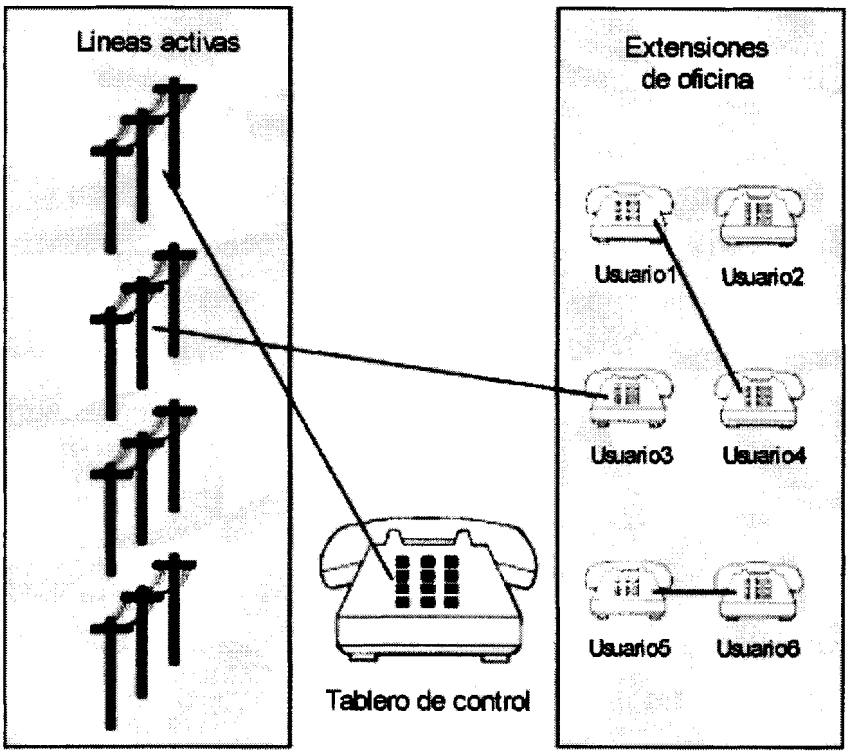

Figura 2.4 Tablero de mando telefónico

2.5 Configuraciones típicas

El modelo TAPI esta diseñado para funcionar en varias Configuraciones físicas diferentes. Cada una tiene sus ventajas y desventajas. Hay cuatro Configuraciones físicas generales:

- La configuración basada en el teléfono: Esta es la más conveniente para la llamada orientada a procesamiento de voz, donde el teléfono estándar (o alguna variación) es usado más frecuentemente.
- La configuración basada en PC: Esta es la más conveniente para la llamada orientada a procesar datos, donde la PC es usada más frecuentemente para procesar voz o datos.

- Línea Compartida o unificada: Este es un compromiso entre sistemas basados en teléfonos y basados en PC´s. Deja a todos los dispositivos manejarse como iguales a lo largo de la línea de servicio.
- Multilínea: Existen varias variaciones de la configuración de multilínea. La diferencia primaria entre esta configuración y las demás son que la PC actúa tanto como un servidor de voz o como una central que conecta las líneas telefónicas exteriores con uno o más PC's y teléfonos.

2.5.1 Configuración basada en teléfonos

En Configuraciones TAPI basadas en teléfonos, el teléfono estándar esta conectado con el conector telefónico y la PC esta conectada con el teléfono (ver Figura No 2.5).

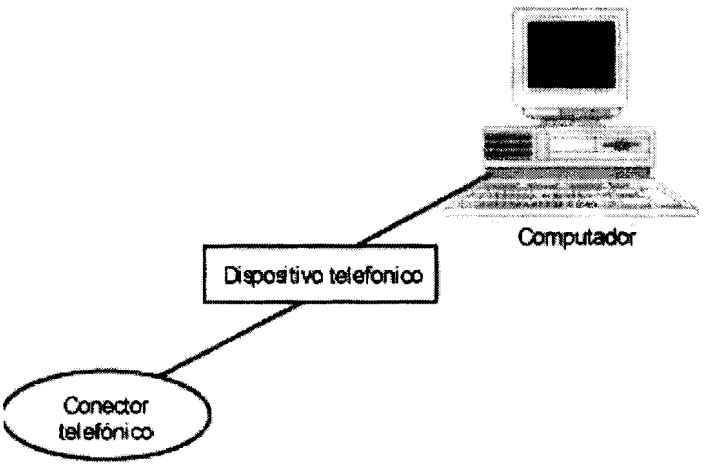

Figura 2.5 Configuración basada en teléfono

Esta configuración es más útil cuando el teléfono es el dispositivo primario para ganar acceso a la línea telefónica. Como el teléfono descansa entre la PC y el conector, la PC no puede participar en toda la actividad sin

retardo. En el ejemplo mostrado en la Figura, la PC no podría ser incluida en una conferencia telefónica si el teléfono originó la llamada, pues ninguna línea puede ser "conferenciada" con si misma, y además, la llamada fue originada "arriba" de la PC.

Una configuración basada en teléfonos no imposibilita el uso de la PC para originar llamadas. Mientras la PC este acondicionada con una tarjeta telefónica (MODEM) que permita llamar, esta puede originar una llamada y entonces permitir el teléfono atender rápidamente esa llamada en cualquier momento. En otras palabras, incluso en una configuración basada en teléfonos, la PC puede ser utilizada como una herramienta de discado, y manejar las llamadas fuera del dispositivo telefónico.

2.5.2 Las Configuraciones basadas en PC's

En las Configuraciones basadas en PC's , TAPI coloca la PC entre el conector telefónico y el teléfono estándar (ver Figura No 2.6).

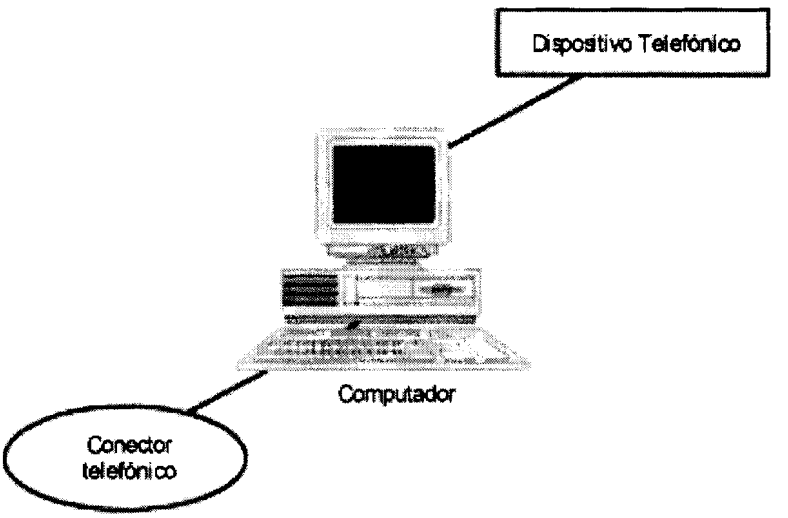

Figura 2.6 Configuración basada en PC

Esta configuración es más útil cuando la PC es el dispositivo primario para ganar acceso a la línea telefónica. En esta configuración, la PC es la que más a menudo origina las llamadas telefónicas. Típicamente, esto esta manejado por una tarjeta telefónica y un software en la PC, el cual utiliza una lista de números de teléfono y maneja el discado del teléfono. Dependiendo del modo exacto de la llamada, la PC puede usarse para mostrar datos digitales en la pantalla mientras se maneja también la información de voz de la llamada.

La configuración basada en PC's también permite originar llamadas desde el teléfono. En este caso, las llamadas pueden ser compartidas por la PC, pues los datos pasan a través de la PC para llegar al conector telefónico. Los usuarios podrían originar una llamada la voz a través del teléfono y después pasar a trabajar en la PC para captura y o mostrar datos digitales expedidos sobre la misma línea.

Otra ventaja principal de la configuración basada en PC's es que la PC puede actuar como una central de llamadas para el teléfono. Esto es especialmente de valor en un ambiente del modo mixto donde, por ejemplo, voz, datos y fax entran por la misma dirección telefónica. Cuando una llamada entra por la línea telefónica, la PC puede responder la llamada y determinar el modo de soporte lógico informático de la llamada. Si es una llamada fax, entonces la PC puede encaminar la información directamente para una máquina a la que se maneja el servicios de fax (un equipo fax o un fax en la PC). Las llamadas de datos pueden ser manejadas directamente por la PC y las llamadas de la voz pueden ser remitidas al teléfono.

En una configuración basada en PC's, la PC también puede servir para mostrar llamadas en pantalla y para el manejo siguiente del mensaje. El software que soporte TAPI podría registrar mensajes entrantes para el

usuario y colocarlos en cola para posteriores llamadas retrospectivas, o simplemente redirigirlas para otra dirección. Con la suma de servicios de identificación de telefónica local (Caller ID), la PC también podría actuar como un agente de filtrado de llamadas, mostrando las llamadas conforme llegan y permitiendo solo a personas predesignadas a acceder a la PC o al teléfono.

2.5.3 Las Configuraciones compartidas y Unificadas de línea

La configuración compartida o unificada de línea es como un compromiso entre Configuraciones basadas en PC's y basadas en teléfonos. La configuración compartida de línea involucra una hacia la línea, conduciendo al conector telefónico. Ambos, la PC y el teléfono, tienen acceso al igual (y simultáneo) a la línea (ver Figura No 2.7).

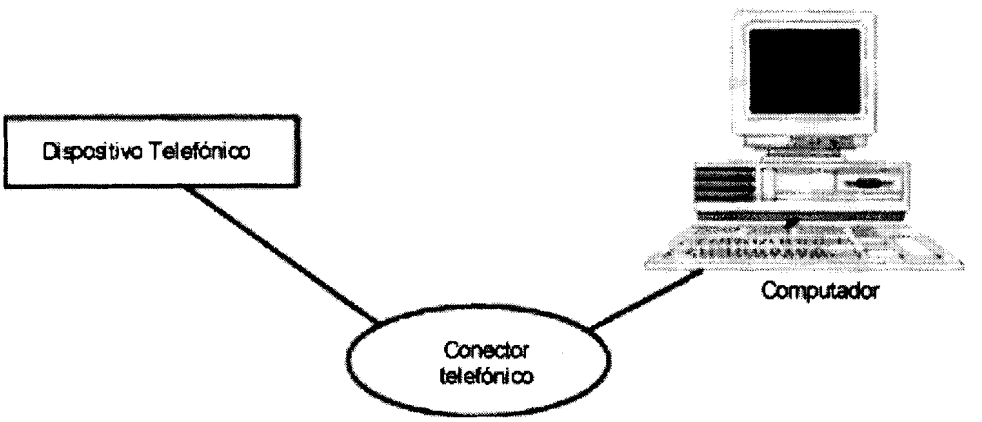

Figura 2.7 Configuración compartida

La ventaja de la configuración de línea compartida es que cada dispositivo puede actuar como el generador de una llamada. La desventaja primaria es que ambos dispositivos tienen acceso por igual a las llamadas entrantes. En otras palabras, cuando una llamada entra, ambos dispositivos

timbrarán. Dependiendo del software operando la PC, es posible que ambos dispositivos intenten satisfacer la misma llamada entrante. Esta situación es como tener dos teléfonos de la extensión a lo largo de la misma línea de acceso.

La configuración unificada de línea ofrece los beneficios combinados de la configuración basada en PC's y la configuración de línea compartida. En la configuración unificada de línea, la línea de acceso va directo del conector telefónico a una tarjeta telefónica (MODEM) en la PC, pero la PC también puede tener el equipo de teléfono ya sea conectada a la tarjeta telefónica o integrado en la PC misma. Todo lo que es realmente necesario es un micrófono para la entrada y parlantes para la salida, pero algunos sistemas ofrece audífonos o un teclado pequeño para simular un teléfono familiar (ver Figura No 2.8).

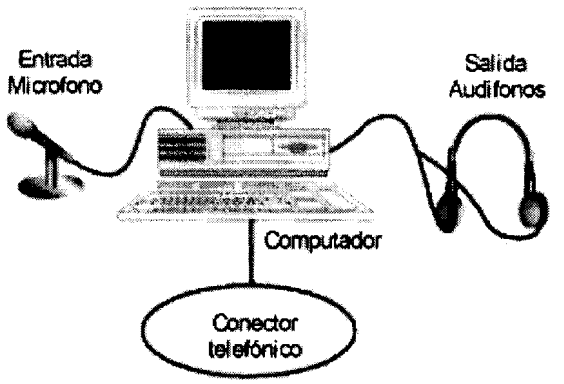

Figura 2.8 Conexiones de la PC´s

Con esta configuración de línea, la PC puede actuar tanto como cualquier dispositivo de teléfono o como un dispositivo de datos de PC. De hecho, la configuración unificada de línea es virtualmente la misma que la configuración basada en PC, excepto que el teléfono es interno para la PC en lugar de estar adjunto a la PC. Como las llamadas nuevas entran

directamente al dispositivo, el software en la PC puede determinar el tipo de medio de la llamada (datos, fax, voz, etcétera) y puede encaminar el requerimiento para el correcto hardware en la PC. También, con el acomodamiento unificado, los usuarios no necesitan preocuparse por el timbrado de dos dispositivos al mismo tiempo cuando una llamada entre al sistema.

2.6 Los servicios asistidos de Telefonía

La forma más simple de servicio TAPI es la de Telefonía Asistida. Bajo la interfaz Asistida de Telefonía, los programadores pueden colocar llamadas con rumbo exterior y pueden monitorear la situación actual del discado desde el mismo puesto de trabajo. Este tipo de servicio de telefonía puede usarse para proveer un simple botón del Dial para aplicaciones ya existentes, o puede añadir capacidades de marcado para aplicaciones nuevas que usarán la telefonía como un servicio añadido.

De hecho, el modelo de Telefonía Asistida solo provee acceso al programa para demandar la colocación de una llamada con rumbo exterior. El discado real de la llamada es manejado por otra aplicación Windows/TAPI. La aplicación predeterminada es DIALER.EXE. Este programa es parte básica de los sistemas operativos Windows.

Hay dos funciones de llamadas de TAPI usadas para proveer servicios Asistidos. La tabla I presente en el anexo A de esta tesis muestra las dos llamadas, sus parámetros, y las descripciones de lo que hacen y como pueden ser usados.

El TAPIRequestMakeCall tiene cuatro parámetros. Solo los DestAddess son requeridos. Los DestAddress son una sucesión de números que representa el número de teléfono para marcar. Por ejemplo, "999-555-1212" es un DestAddress válido en el formato de Estados Unidos. El parámetro AppName es el nombre de la aplicación que demandó el servicio TAPI. El CalledParty es una variable que representa el nombre de la persona a quien se llama. Esta información podría ser usada por la aplicación del DIALER.EXE para poner en un historial a la persona llamada. El parámetro del Comentario podría contener una variable de texto describiendo la razón para la llamada.

La función TAPIGetLocation devuelve dos parámetros: El CountryCode y CityCode, siendo estos determinados por el Control Panel de Windows. Estos dos parámetros son almacenados en el archivo del TELEPHON.INI en la carpeta de Windows. El código del país es un valor usado para colocar llamadas fuera de país. El CityCode es conocido como el código de área. La combinación del código del país y el código de la ciudad se usa para determinar como colocará el marcador TAPI la llamada demandada.

Por ejemplo, si la llamada demandada contiene " 1-312-555121", entonces y la localización actual del puesto de trabajo indicaban un código del país de "1" y un código de la ciudad de "312," .El programa TAPI DIALER.EXE intentara colocar la llamada sin incluir el país o códigos de la ciudad: "555-121"" Si, sin embargo, la llamada demandada contenía "43-80-12 33 45", entonces el programa DIALER.EXE da por supuesto que el usuario trata de colocar una llamada fuera de país y usaba los prefijos apropiados de discado.

2.7 Los servicios básicos de Telefonía

La telefonía básica es el siguiente nivel levantado en el modelo de servicios TAPI. Las llamadas básicas de función de Telefonía permiten a los programadores crear aplicaciones que puedan proveer manejo básico de llamadas de voz y datos tanto con rumbo exterior como entrantes, sobre un teléfono analógico de la sola línea. La línea telefónica analógica usada más a menudo para este nivel de servicio es la POTS . Este conjunto de funciones Básico de API de Telefonía también puede ser usado con más líneas sofisticadas como T1, ISDN, o líneas digitales. Sin embargo, las características añadidas de estos dispositivos mas adelantados de línea (como reenvío, parqueo, conferencia, etcétera) no están disponibles al usar la API Básica de Telefonía.

El servicio de Telefonía Básica enfoca la atención en el uso de un dispositivo de línea como una manera de transportar información de un lugar para otro. Un dispositivo de línea para TAPI puede ser un teléfono, un fax, un módem de datos, una tarjeta de telefonía, o cualquier otro dispositivo físico que puede estar conectado a una línea telefónica. Sin embargo, éste es tratado como un dispositivo virtual, no uno físico.

Los dispositivos de línea no son asociados directamente con cualquier línea telefónica física. Así, TAPI "puede ver" dispositivos múltiples en la misma máquina (el módem de datos, el teléfono, y el fax) mientras que solo se tiene una sola línea telefónica física real conectada con el puesto de trabajo (ver Figura No 2.9).

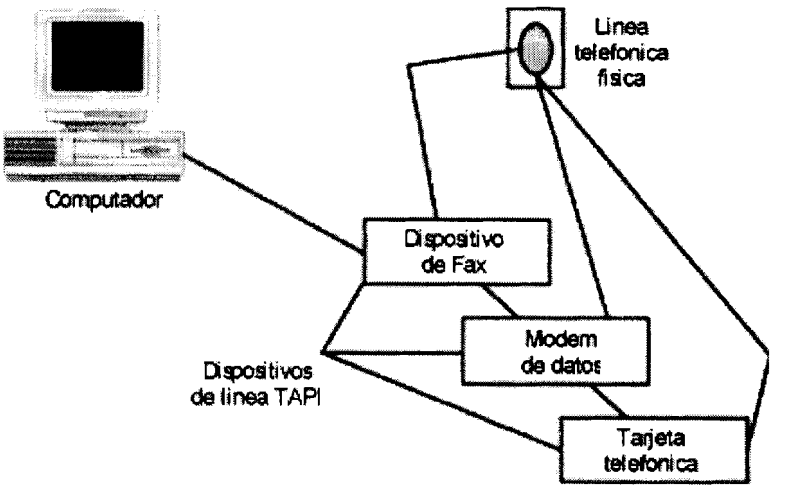

Figura 2.9 Dispositivos de línea

Una de las funciones primarias de la interfaz TAPI es manejar múltiples requerimientos de servicio TAPI del puesto de trabajo. Esto es posible, por ejemplo, cuando varias existen varias aplicaciones funcionando en el puesto de trabajo, cada una de las cuales pueden demandar servicios TAPI en algún tiempo. La aplicación de control de llamada (DIALER.EXE) acepta cada petición y los encola según prioridades para procesar en su debido tiempo la orden demandada.

2.7.1 El conjunto de funciones del Dispositivo básico de Telefonía

El modelo Básico de servicio de Telefonía tiene varias llamadas de API para manejar y cumplir a cabalidad las múltiples demandas de servicio. Estas llamadas pueden ser distribuidas en grupos lógicos:

- Las llamadas de manejo básicas de línea, manejan la inicialización, la apertura y el cierre de líneas TAPI.

- La configuración y el estado de la línea, manejan la lectura y escritura de diversos parámetros que controlan el comportamiento del dispositivo de línea.
- Las funciones de rumbo exterior y de entrada, manejan los detalles de colocar una llamada de voz o de datos con rumbo exterior y responder una llamada de voz o de datos de entrada.
- Funciones que dirección, manejan los detalles de reconocer, traducir, y/o construir instrumentos de "direcciones" telefónicas o de discado.
- Las características misceláneas, manejan otras funciones relacionadas a TAPI, como manejar privilegios de monitorear llamadas y manipular el manejador de la llamada.

La tabla II presente en el anexo A de esta tesis muestra las llamadas Básicas de API de Telefonía, clasificadas por el grupo funcional, junto con una descripción pequeña de su uso.

2.7.2 La estructura básica del Dispositivo de línea telefónica

Junto con el extenso conjunto de funciones de API para la Telefonía Básica, el modelo TAPI define varias estructuras de datos que se usan para pasar información entre TAPI y la aplicación peticionaria. El trazado de las estructuras contiene tanto variables como también datos fijos. Esto permite al API contener información de longitud indeterminada sin anterior conocimiento del contenido de la estructura.

Para manejar estructuras de longitud variable, las estructuras definidas de datos contienen campos que indican que tamaño total necesitó para suplir todos los datos variables (dwNeededSize) junto con el tamaño total usada

por TAPI para suplir la estructura (dwUsedSize). El siguiente listado muestra como esto se ve en la estructura LINECALLLIST.

```
typedef struct linecalllist {
  DWORD dwTotalSize;
  DWORD dwNeededSize;
  DWORD dwUsedSize;

  DWORD dwCallsNumEntries;
  DWORD dwCallsSize;
  DWORD dwCallsOffset;
} LINECALLLIST, FAR * LPLINECALLLIST;
```

El campo dwTotalSize es determinado por la aplicación de llamado decir cuánta memoria ha sido ubicada para la estructura. Si TAPI no puede suplir todos los valores sin quedarse sin espacio ubicado, entonces un error es devuelto y es trabajo de la aplicación peticionaria reasignar espacio y hacer la llamada otra vez.

Junto con el tamaño y el total necesario de campos, cada estructura de longitud variable tiene una porción fija y una porción variable. La porción fija contiene valores que indican el tamaño del campo de longitud variable y el offset (distancia desde el principio de la estructura) en el cual el campo esta ubicado. Note los campos dwCallsSize y dwCallsOffset en la estructura LINECALLLIST demostrada en la lista.

La tabla III presente en el anexo A de esta tesis muestra la lista de estructuras de datos usada por la API Básica de Telefonía junto con descripciones básicas de su uso.

2.7.3 Los mensajes básicos del Dispositivo de línea telefónica.

El Telefonía API usa los mensajes de Windows para comunicarse con la aplicación peticionaria. Cuando la aplicación realice la función de inicio LineInitialize, la dirección de una función de respuesta-recuperación debe ser suministrada (callback). Todos los mensajes son entonces expedidos para esta función única de callback.

El hecho que TAPI use callbacks para la recuperación de los mensajes significa que cualquier lenguaje de alto nivel, como Basic Visual, debe usar ya sea un DLL u OCX, o debe establecer un eslabón de comunicación con cualquier herramienta que pueda capturar los mensajes de Windows.

Cada mensaje devuelve el mismo grupo de parámetros. La primera parte es el manejador relevante. Usualmente este es el manejador de la llamada, pero también puede ser el simple manejador de la línea, lo cual no es lo mismo. El segundo parámetro es el valor de la instancia del callback. Este valor siempre será el manejador del progreso de la aplicación actual que esta corriendo, y es el que determinara el estado del progreso de las peticiones. Los siguientes tres valores varían según la naturaleza del mensaje. Uno o más de estos valores de regreso contendrán datos que no sean cero. La tabla IV presente en el anexo A de esta tesis contiene una lista de los mensajes Básicos de Telefonía, sus parámetros, y sus descripciones pequeñas.

2.8 Los servicios suplementarios de Telefonía

Las funciones suplementarias de telefonía proveen manejo avanzado (conferencia, parque, agarre, reenvió, etcétera) del dispositivo de línea. El

acceso para estos servicios adelantados está bajo la dependencia del tipo de línea telefónica en el cual el puesto de trabajo está conectado. En otras palabras, aun si se implementa funciones avanzadas dentro de su aplicación TAPI, estas funciones solo operarán si estos servicios están disponibles en la línea telefónica proveída por la compañía telefónica local.

Las funciones suplementarias de telefonía también permiten a los programadores manejar demandas múltiples de servicio para teléfonos de línea múltiple. Se puede usar la Telefonía Suplementaria para manejar un teléfono físico multilínea, esto es, que tiene acceso a varias líneas físicas múltiples (ver Figura No 2.10).

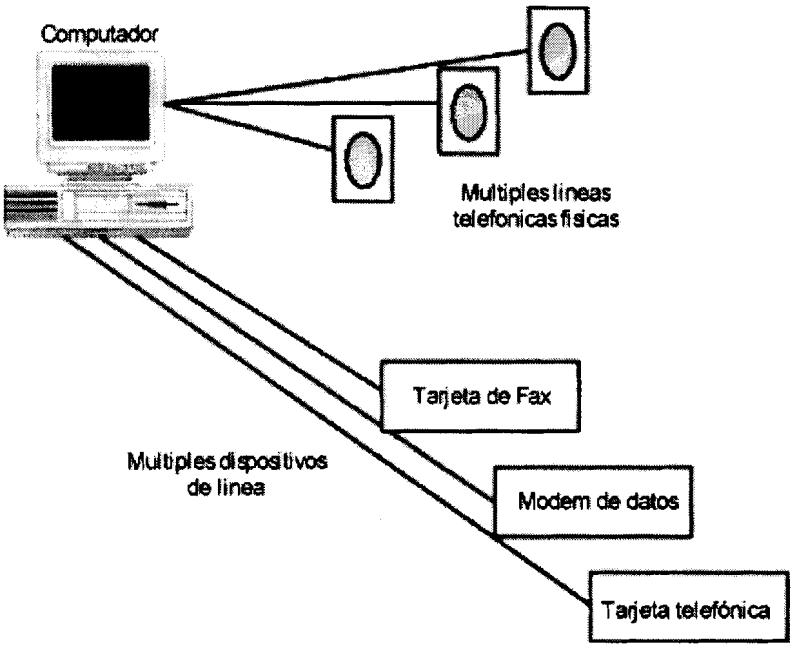

Figura 2.10 Manejo de demandas múltiples

También se puede usar las funciones suplementarias de telefonía para operar varios teléfonos múltiples usando a una o más líneas físicas. Debido a que TAPI "virtualiza" los dispositivos telefónicos y de línea, no hay necesidad

de una correspondencia individual directa entre un dispositivo telefónico definido y un dispositivo definido de línea. De este modo se puede usar a TAPI para crear una aplicación de tablero de mando para manejar servicios de telefonía (ver Figura No 2.11)

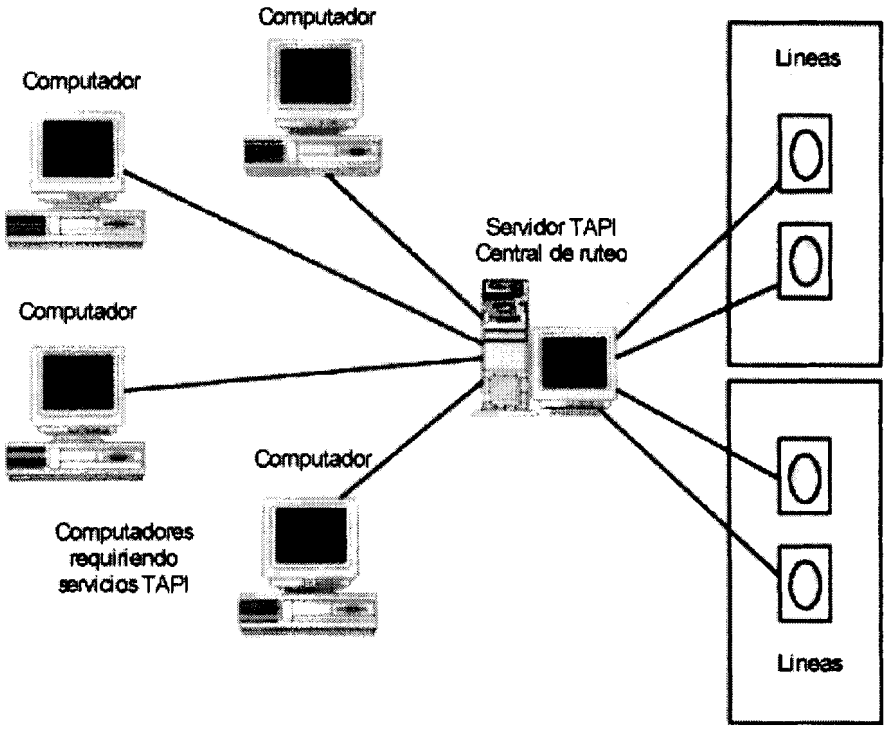

Figura 2.11 Servidor de tablero de mandos

La telefonía suplementaria también provee acceso para definir y manipular dispositivos telefónicos. Para TAPI un dispositivo telefónico es cualquier dispositivo que puede aceptar o realizar una llamada. En efecto, se puede registrar un puesto de trabajo como un dispositivo telefónico. Entonces se podrá usar los recursos en el puesto de trabajo para colocar o aceptar llamadas sin la necesidad de un teléfono. Por supuesto, para actuar exitosamente como un dispositivo telefónico, el puesto de trabajo debe tener hardware de entrada y de salida de audio.

2.8.1 El API suplementario de telefonía para dispositivos de línea

El conjunto de funciones Suplementario de API para dispositivos de línea añade control avanzado de llamada y otras características para la biblioteca de API. El conjunto de funciones puede estar dividido en los siguientes grupos relacionados de funciones:

- Manejo de funciones de dígito y tono permiten a los programadores detectar y generar dígitos o tonos a lo largo de línea telefónica. Esta capacidad es necesaria para permitir a algunos sistemas realizar operaciones avanzadas de línea como reenvío, espera de llamada, etcétera.
- Funciones avanzadas de manejadores de línea proveen aceptación de llamada, rechazo, reexpedición, y otras operaciones. Estos son más útiles en un ambiente donde la línea telefónica está conectada con un interruptor central en lugar de directamente con la línea telefónica externa.
- Funciones avanzadas de llamada actual proveen "Agarre de Llamada", Transferencia, reenvío, y capacidades de respuesta. Estas funciones sólo surten efecto si la línea telefónica respalda estos requerimientos avanzados.
- Las funciones avanzadas de características misceláneas proveen características añadidas específicas que el servicio demanda a TAPI, como monitorear líneas y colocar parámetros de llamada.

La tabla V presente en el anexo A de esta tesis muestra todas las funciones Suplementarias del API de Telefonía para las características avanzadas del dispositivo de línea.

2.9 Los servicios extendidos de Telefonía

El último nivel de servicios de Telefonía es Telefonía Extendida. El servicio extendido de Telefonía permite a los vendedores-proveedores del hardware definir funciones específicas en dispositivo y los servicios propios, sin impedir que sigan funcionando bajo el modelo de servicio TAPI. Añadiendo un grupo pequeño de funciones extendidas de API de servicio, Microsoft permite a los vendedores-proveedores del hardware continuar proveyendo servicios únicos no previamente definidos por TAPI. El modelo TAPI aun define tanto la línea y los llamados del dispositivo para la Telefonía Extendida. La tabla VI presente en el anexo A de esta tesis muestra el conjunto de funciones Extendidas de API de Telefonía junto con descripciones pequeñas de su uso.

El significado real y el uso de llamadas extendidas TAPI están definidos por el proveedor de servicios o el vendedor-proveedor del hardware. Los proveedores extendidos de Telefonía definen los parámetros de las llamadas y su significado, y publican esta información para el programador. El programador entonces puede cotejar la información de versión con el país proveedor de servicios antes de intentar hacer una llamada extendida de servicio.

2.10 Resumen

La Interfaz de Programación de Aplicaciones de Telefonía (TAPI) es uno de los conjuntos de funciones de API más significantes liberados por Microsoft. El API de telefonía es un conjunto de funciones determinado de llamadas de función que permite a los programadores manipular cualquier tipo de conexión de comunicación entre la PC y la línea telefónica. Mientras

los modelos de telefonía para PC han existido por varios años, la API de telefonía establece un grupo uniforme de llamadas que puede ser aplicado a cualquier tipo de hardware, siempre y cuando esta suministre una interfaz condescendiente a TAPI por parte del proveedor de servicios.

TAPI logra su tarea dividiendo el trabajo en dos estratos distintos: El cliente API y el SPI. Cada uno interactúa con un grupo de funciones diseñadas para realiza tareas genéricas de telefonía, como abrir una línea, monitorear el tono de marcar, marcar un número, comprobar un ring o una señal de ocupado, etcétera. La API del cliente envía las demandas múltiples de la aplicación al SPI para cada tarea. Es trabajo del SPI completar la tarea y enviar los resultados para el programa de llamado a través del cliente API.

El modelo del diseño TAPI está dividido en dos áreas, cada uno con su propio conjunto de funciones de llamadas de API. Cada conjunto de funciones de API enfoca su atención en lo que TAPI se refiere como a un dispositivo. Los dos dispositivos TAPI son:

- Los dispositivos de la línea: modelan las líneas físicas de telefonía a usarse para enviar y recibir voz y datos entre localizaciones.
- Los dispositivos telefónicos: modelan el medio físico (el teléfono de mesa) usado para colocar y recibir llamadas.

Capítulo 3

Caller ID

Dependiendo del lugar en donde se encuentre, los formatos de datos de Caller Id pueden variar en gran medida. Es así que dependiendo de la localidad pueden existir diferentes maneras de poder utilizar la trama de Caller Id. Para ver a la información de CALLER ID primeramente se necesita el servicio de CALLER ID del proveedor de telecomunicaciones, lo cual quiere decir que los datos de CALLER ID deben ser enviados en el formato soportado por su hardware. Además de ello se debe tener un equipo que pueda manejar esa trama de información Caller Id. Estos equipos pueden ser uno de estos:

- Un teléfono adecuado de reconocimiento de CALLER ID.
- Una máquina de reconocimiento de CALLER ID.
- Una tarjeta de identificación de Caller Id dedicada en su computadora con el software adecuado.
- Un modem/ISDN que reconozca los datos de CALLER ID con software adecuado.

Se comenzará analizando los fundamentos arriba descritos sobre CALLER ID en los diferentes sistemas, como un teléfono o caja de Caller Id.

3.1. Servicio de CALLER ID

En los días de las comunicaciones analógicas, la Identificación Automática de Número (ANI) fue desarrollada para propósitos de facturación de llamadas. La información nunca traspasó el la localización de la persona que llama, excepto para esas personas con conexiones privilegiadas para la red como el servicio del 999/112/911, el policía, y, más tarde, los proveedores de servicio gratuito. Estos grupos continúan usando a ANI que está completamente separado de CALLER ID y no pueden ser bloqueados.

El servicio de caller id fue posible gracias a la adopción mundial de sistemas de intercambios digitales de datos. Esto se realiza habilitando circuitos de datos separados para los involucrados en el intercambio. Básicamente, para que el servicio de Caller Id funcione debe realizar un intercambio digital de datos entre los dos usuarios conectados, el cual se realiza generalmente utilizando el sistema ss7 (signalling system 7). SS7 es actualmente el sistema mundial de conexión de las compañías telefónicas. SS7 permite al intercambio de datos de los usuarios enviar el paquete denominado CPNM (calling party number message) el cual incluye el número del usuario y el indicador de bloqueo del número de este. CPNM funciona simplemente pasando la información del número y del bit de bloqueo (o privacidad) al usuario llamado. El trabajo de realizar la traducción de la información se realiza localmente, para luego ser presentado o bloqueado según indique el bit de privacidad del paquete de datos.

Actualmente muchas compañías telefónicas proveen en su totalidad conexiones ss7, pero no correctamente habilitadas en todo su sistema telefónico. Incluso el exceso de ruteo en centrales mal administradas puede hacer que el intercambio de números de caller id no pueda ser terminado satisfactoriamente. Además, como todo intercambio de dato especializado,

normalmente no se puede proporcionar el servicios de caller id en líneas analógicas. A pesar de ser posible en los sistemas telefónicos, el servicio de CALLER ID necesita ser pedido y habilitado antes de que se lo pueda usar.

3.2. Las normas - estándares de CALLER ID

El estándar aplicado en cada país no es necesariamente el mismo. Existen varios tipos de estándares para el proceso de mandar el paquete de Caller Id, y normalmente dependen de los equipos centrales utilizados en cada localidad.

El estándar Bellcore es usado en la USA, Canadá, China, Australia, Italia, etc, y es el más normalizado de todos, por lo que será el que principalmente analizaremos. Este sistema envía los datos después del primer tono del ring y utiliza la modulación de 1200 baudios - Bell de 202 tonos. Los datos pueden ser enviados en "Simple Formato de Mensaje de Datos" (SDMF) que incluye la fecha, la hora y número, o puede mandar el paquete de información en el tipo de "Formato Múltiple de Mensajes de Datos" (MDMF) que agrega un campo de NOMBRE.

Los aspectos originales para Caller ID toman en consideración el ser enviados tanto el nombre de la persona que llama como tanto como el número. Aunque este servicio (mandar nombres) no esta comúnmente implementado. Sin embargo, existen soluciones que hacen transparente este problema. Lo mejor que se puede hacer es tomar el número entrante y compararlo con su libro de direcciones o identificadores de números telefónicos - los dispositivos externos de caller id han limitado memorias para hacer este tipo de cosa, las soluciones de la computadora obviamente pueden hacer mucho más con estos datos.

El denominado Entrega de Número Llamante (CND), es el servicio telefónico desarrollado para clientes residenciales y de negocio pequeño por el sistema Bellcore. Este permite al denominado "Equipo de Premisas del Cliente" (CPD) recibir un número, la fecha y la hora de la llamada por parte del usuario remoto durante los primeros cuatro segundos del intervalo silencioso en el ciclo de llamado. El cliente debe contactar a una compañía para iniciar el servicio CND, pues no esta preestablecido.

3.3. Parámetros y Características

La interfaz de la señal de datos tiene las siguientes características:

- El tipo del eslabón: Simples, de 2 alambres
- El esquema de transmisión: MULTIPLEXACIÓN DE DIVISIÓN DE FRECUENCIA analógica, coherente en la fase(FSK)
- Lógico 1 (marca): 1200 + / - 12 Hz
- Lógico 0 (espacio): 2200 + / - 22 Hz
- Tasa de transmisión: 1200 bits por segundo
- El nivel de transmisión: 13.5 + / - dBm en carga de 900 ohms

3.4. Protocolo

El protocolo usa palabras de datos (los bytes) de 8 bits, cada uno definido por un bit de principio y uno de detener. El mensaje CND usa el Formato Simple de Mensaje de Datos mostrado abajo en la figura No 3.1.

1	2	3	4	5
Señal de Agarre del Canal	Señal de Transportador	Tipo de Mensaje	Largo de Palabras Mensaje de Datos	Suma de verificación

Figura 3.1 Trama Caller Id

La figura No 3.2 muestra visualmente la asociación del primer ring con la señal de agarre del canal, la señal del transportador, la información Caller Id, la suma de chequeo y el segundo ring.

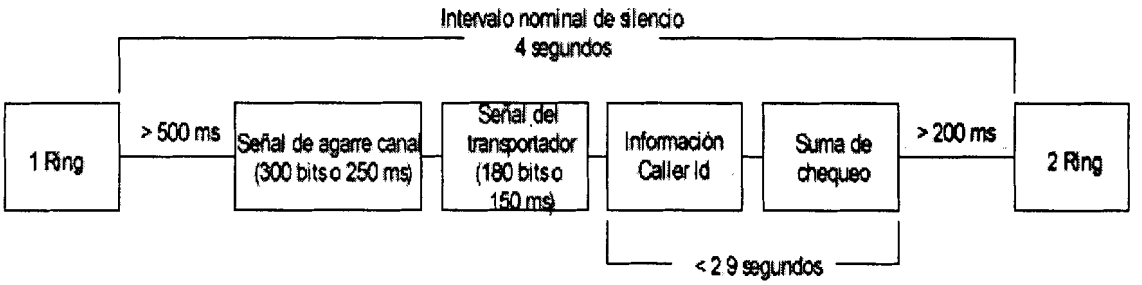

Figura 3.2 Relación de paquetes Caller Id y Tiempos

3.4.1. Señal de Agarre del Canal

El agarre del canal son 30 bytes continuos de 55H (01010101) de tal manera que sea una función alternante detectable para el EPC (funciona como datos de iniciación del módem).

3.4.2. Señal de portadora

La señal portadora consta de 130 + / - 25 mS de marca (1200 Hz) para guiar al receptor para datos.

3.4.3. Tipo de Mensaje

La palabra de tipo de mensaje indica que el servicio y la capacidad asociada con el mensaje de datos. El tipo de mensaje para CND es 04h (00000100).

3.4.4. Largo de Mensaje e información del campo Mensaje de Datos

La palabra de largo de mensaje especifica el número total de palabras de datos a seguirse en el campo siguiente, el de Mensaje de datos, el cual representa la información Caller Id en si. La información de las palabras del campo Mensaje de Datos está codificado en ASCII y tiene un solo tipo de significado característico. La información presentada por las palabras de datos es la siguiente:

- Las primeras dos palabras representan el mes
- Las siguientes dos palabras representan el día del mes
- Las siguientes dos palabras representan la hora en el tiempo militar local
- Las siguientes dos palabras representan el minuto después de la hora

El número telefónico del usuario que llama es representado por las palabras remanentes en el campo de Mensaje de Datos. Si el número no está disponible para la oficina central que termina, entonces el campo de palabra de datos contiene un ASCII "O". Si el usuario que llama invoca al bit de intimidad, entonces el campo de palabra de datos contiene un ASCII "P".

3.4.5. Palabra Suma de verificación

La palabra suma de verificación contiene el complemento a 2 del módulo de suma 256 de las otras palabras en el mensaje de datos (i.e., El largo del tipo mensaje, y las palabras de datos). El equipo receptor puede calcular al módulo suma 256 de las palabras admitidas y puede añadir esta suma para la verificación de los datos recibidos. Un resultado de cero generalmente señala que el mensaje fue correctamente admitido por la mayoría. La retransmisión de mensaje no es respaldada.

3.4.6. Ejemplo de Mensaje de Datos Simple CND

Un ejemplo de un mensaje admitido por la mayoría de los CND, comenzando desde la palabra de tipo de mensaje:

El teléfono obtiene el primer ring, con duración de 2 segundos. Un retraso 0.5 segundos prosigue al primer ring. Se envía 30 bytes de "01010101", con una duración de 250ms (señal de agarre del canal). Se envía 150ms de marcas (la señal del transportador). Se envíe un "parámetro" de 8 bits, el cual es el indicador de tipo de palabra. Estos parámetros pueden ser:

00000100 = Caller Id
00001010 = Indicador de mensajes de espera
10000001 = Prueba para la identificación del Caller id

Se prosigue enviando un número de 8 bits representando el número de PALABRAS en el mensaje, para finalmente enviar los datos reales. Los últimos 8 bits del mensaje son la suma de verificación. La suma de

comprobación es el complemento de dos del modulo 256 suma de las otras palabras en el mensaje de datos. Esta suma no incluye la señal de agarre del canal o la señal del transportador.

El total de duración de la transmisión ha sido de 718ms, y el mensaje obtenido llega en el formato siguiente:

04 12 30 39 33 30 31 32 32 34 36 30 39 35 35 35 31 32 31 32 51

04h : tipo de mensaje, en este caso, tipo palabra
12h = 18 decimal: Número de ASCII de palabras de datos (la fecha, tiempo, y el número palabras párale número telefónico)
30,39 = 09: El ASCII de septiembre
33,30 = 30: El ASCII de día 30
31,32 = 12: 12:00 PM ASCII
32,34 = 24: ASCII de 24 minutos (12:24 PM)
36,30,39,35,35,35,31,32,31,32 = (609) 555-1212: Número telefónico
51h: Suma de verificación del número

3.5.Caller Id en PC's

Ante todo, los módems debes poder seguir ciertos requerimientos. Lo que sucede al nivel técnico es lo siguiente. Aunque los datos dados sean señales de parámetros de la interfaz del tipo Bell 202 módem, el receptor no necesita ser un módem Bell 202. A partir de V.23 1200, el receptor del módem ya puede estar normalmente habilitado para demodular la señal Bell 202. El bit de indicación de timbrado (RI) puede ser usado en un módem para indicar cuando monitorear la línea telefónica para la información CND. Después del set de bits de RI, indicando el primer ring, se espera para el RI

de reseteo. Es aquí entonces cuando se configura el módem para monitorear la línea telefónica para la información CND.

Normalmente, el software recupera el Caller Id abriendo el Puerto Com y esperando a que el hardware telefónico envíe los paquetes de datos cuando se detecta y conecta un llamado externo. El problema con esto es que ningún otro programa puede acceder y utilizar los recursos del puerto Com hasta que el programa de Caller id obtenga los datos que desea y desocupe el puerto. Esto es un inconveniente importante si ocurre en máquinas que deben correr procesos múltiples con una sola salida de línea, como estar conectados a una red y ser un programa telefónico al mismo tiempo. Por esta razón se introdujo el conjunto de funciones TAPI. Este actúa como un intermediario de acceso para el módem para todo programa que quiera usarlo. Esto le permite usar el acceso telefónico para enlazar una red sin cerrar el programa de Caller Id, si su programa de la CALLER ID es compatible a TAPI. Así es que es una buena idea asegurarse que su programa de la CALLER ID es condescendiente con TAPI.

TAPI 1.3/1.4 fue una versión híbrida de 16bit/32bit que vino con Windows 95; NT 4.0 introdujo el TAPI de 32 bits con la versión 2.0 y 2.1. La telefonía de Novell, la API del Servidor, TSAPI, es similar, pero sólo corre en Netware. Sin embargo, es algo similar en funcionamiento a TAPI, pero no alcanza su desarrollo.

A manera de una introducción breve, hay dos formas de usar TAPI. Para obtener Caller Id. En general, una llamada entrante hará al TSP generar un mensaje de la Line _ CALLINFO, entonces un llamado a lineGetCallInfo () debería conducir hacia los campos de CALLER ID inmersos en la llamada.

3.5.1. CALLER ID en Windows 95

Muy pocos módems pueden entender más que un estándar. En Windows 95 se deben realizar algunas adecuaciones especiales para habilitar este servicio. Para que TAPI trabaje, necesita un tipo especial de driver de módem conocido como "proveedor de servicios de telefonía", o TSP. Windows 95 y 98 ya poseen un TSP genérico llamado Unimodem/V que apareció desde el OSR2. Esto añade características de voz ausentes en el anterior Unimodem. Se puede listar los TSPs en el sistema habilitando el applet de Telefonía en el Panel de Control: renombrando el archivo TELEPHON.CP$ en C:\WINDOWS\SYSTEM \ a TELEPHON.CPL,. El applet ya esta habilitado en Win98 y NT 4.0.

3.5.2. CALLER ID en Windows 98

Aunque Unimodem/V está teóricamente habilitado y probado, es conocido que aun existen problemas con CALLER ID en Windows 98. La razón principal es que los principales drivers de Windows 98 del tipo "Plug'n'Play" (especialmente para 3Com/USRs) no soportaron una parte de funciones, así es que lo mejor es siempre conseguir los últimos drivers y realizar la actualización de ellos en el Panel de Control.

3.6. Resumen

La información Caller Id es transmitida utilizando tonos de MODEM con modulación de frecuencia (FSK). Estos tonos son del tipo de modulación, frecuencia y formato de datos del sistema Bell 202, y usados para transmitir el mensaje utilizando directamente el código ASCII. Este envío de paquetes de información es ejecutado entre el primer tono de ring y el segundo, siendo

la información incluida el de fecha, hora y número telefónico, siguiendo el formato SDMF (formato de mensaje de datos simples). En algunos casos, el nombre asociado con el número telefónico también es enviado, siguiendo el formato MDMF (formato de mensaje de datos múltiples).

Para obtener Caller Id usando TAPI, en general una llamada entrante hará al TSP generar un mensaje de la LINE _ CALLINFO, entonces un llamado a lineGetCallInfo () debería conducir hacia los campos de CALLER ID inmersos en la llamada. Estos después (dependiendo de las capacidades del MODEM) deberán ser descodificados, utilizando el largo conocido de la palabra y el formato (ASCII).

Capítulo 4

Servicios extendidos de TAPI

Hasta ahora, todas las configuraciones revisadas aquí han sido de modelos de solo una línea. Sin embargo TAPI se compone de varias otras partes en su modelo. Estos son básicamente servicios extendidos para otros dispositivos, los cuales no corresponden a lo utilizado por nuestro programa telefónico. Sin embargo estas funciones serán presentadas y explicadas para exponer al modelo TAPI en su totalidad, pues muestra su valor como dispositivo de desarrollo extendido de aplicaciones.

TAPI esta diseñado también para respaldar configuraciones de multilínea. En este modelo, TAPI se usa para proveer lo denominado control de llamada de terceros. Parte importante de este capítulo corresponderá al estudio de los dispositivos telefónicos.

Las líneas simples actúan como la primera y única parte en la llamada telefónica. En un ambiente de multilínea, un dispositivo puede actuar e intervenir como un tercero en una llamada telefónica. La forma más común de controlar intervenciones de terceros es mediante un tablero central de mando en una oficina. Cuando una llamada entra, el tablero de mando (el tercero) acepta la llamada, determina el destino final de la llamada, encamina el requerimiento para la extensión correcta, y finalmente se desvincula fuera del flujo de comunicación.

4.1 Configuraciones Multilínea

Estos son las dos configuraciones TAPI básicas de multilínea:

- Servidor de voz: Se usa para proveer mensajería de voz y otros servicios de una localización central.
- Servidor Pbx: Se usa para proveer control de llamadas entrantes y salientes para líneas interurbanas.

En una configuración de servidor de voz, la PC actúa como un dispositivo de almacenamiento de mensajes accesibles a cualquier otro elemento telefónico. Los teléfonos pueden ser configurados para remitir todos los mensajes "ring / sin respuesta" al servidor de voz, donde las personas que generan las llamadas reciben indicaciones para dejar mensajes grabados. Más tarde, los usuarios pueden marcar por teléfono en el servidor de voz para recuperar sus mensajes. Alternativamente, los usuarios podrían consultar un inbox que contenga el correo de voz, faxes, y/o correo electrónico. Al tiempo de seleccionar los artículos del correo de voz, la PC reproduciría el mensaje en los altavoces de la PC y permitiría al usuario responder usando ya sea el elemento telefónico o un micrófono conectado.

En una configuración de servidor de PBX, la PC actúa como un tipo de la primera estación para todas las llamadas entrantes para la localización multilínea, usualmente un bloque de oficinas. En esta situación, las funciones TAPI se usan para aceptar llamadas, presentarlas a un operador para su revisión, y remitirlas al destino final.

Mientras el Servidor de PBX hace su trabajo delante de cualquier elemento teléfono, un Servidor de voz hace su trabajo oculto detrás de un elemento telefónico estándar. En otras palabras, el Servidor de voz es diseñado para manejar llamadas cuando el sistema telefónico esta ocupado o en cualquier situación que lo deje incapaz de aceptar llamadas. El Servidor de PBX, por otra parte, esta pensado para aceptar todas las llamadas entrantes a la oficina y enrrutarlas para el teléfono de mesa apropiado. Muchas oficinas utilizan ambos Servidor de PBX y los sistemas del Servidor de voz. El Servidor de PBX responde la petición entrante y la encamina al teléfono apropiado. Si el teléfono es incapaz de aceptar la llamada, el Servidor de voz toma el mensaje y lo almacena para su posterior recuperación.

4.2 Los servicios de línea telefónica

Los tipos de línea pueden estar divididos en tres grupos principales:

- Las líneas analógicas
- Las líneas digitales
- Las líneas de protocolos privados

Las líneas analógicas son el tipo de líneas disponibles en la mayoría de casas. Las líneas digitales son usualmente usadas por organizaciones grandes, incluyendo proveedores de servicios telefónicos locales, para transferir cantidades grandes de canales de voz y de datos. T1 y líneas de la ISDN son tipos típicos de líneas digitales. Las líneas de protocolo privado son tipos especiales de líneas digitales. Estas líneas son usadas dentro de los intercambios privados de sucursales (PBXs). Las líneas del tipo PBX se usan para transportar voz, datos, e información de control especial usada por el

hardware conmutativo para proveer características avanzadas de telefonía como la transferencia de llamada, teleconferencia, etcétera.

4.3 La Red Telefónica

A pesar del tipo de servicio de línea telefónica usada (POTS, T1, ISDN, PBX), la transmisión (voz o datos) debe moverse de la fuente (el origen de la llamada) hacia el destino deseado. Por el camino, una llamada típica puede convertirse de analógico a digital, pasar de alambres físicos a fibra óptica, y posiblemente también pasar a manera una transmisión de satélite antes de que finalmente llegue a la dirección de destino. Proseguiremos describiendo cada uno de los tipos de servicio de línea telefónica en mayor detalle.

4.3.1 El servicio POTS (Plain Old Telephone Service)

El servicio POTS es el tipo de servicio que se provee para la mayoría de casas en los países. Los POTS es un servicio analógico que provee una conexión básica hacia la central telefónica a manera de una conexión de la sola línea. Los usuarios POTS estándar no pueden realizar operaciones telefónicas avanzadas como las transferencias de llamada, enviando, o la teleconferencia.

Los POTS analógicos son diseñados para enviar señales de voz, no datos. Por esta razón, los usuarios POTS deben utilizar un modulador-demodulador (o MODEM) para enviar información digital sobre líneas POTS. Los aspectos analógicos de los POTS limitan la cantidad de datos digitales que se puede proporcionar a través de la línea. Mientras el servicio de

28.8Kbps sobre estas líneas es realmente fidedigno, las tasas de datos más allá de ese límite requieren condicionamiento especial de la línea.

Recientemente, las compañías telefónicas locales han comenzado a ofrecer servicios adicionales a los usuarios. El CALLER ID, el reenvío, y los servicios de mensajería de voz pueden ser comprados de operadores locales por un cargo adicional.

4.3.2 Las líneas digitales del T1

Las líneas digitales T1 son diseñadas para transportar varias conversaciones al mismo tiempo. Las líneas T1 pueden enviar 24 conexiones múltiples. Desde que las líneas T1 son digitales en lugar de líneas analógicas de datos, puede extenderse mucho más allá del alcance de las líneas analógicas. Las tasas de datos de millones de caracteres por minuto son típicas para las líneas dedicadas T1. Las líneas T1 sirven típicamente para transmisiones dedicadas de datos y transmisión de conversaciones múltiples de punto a punto.

Las líneas telefónicas europeas tienen un formato similar para el T1, llamado E1. Las líneas E1 pueden manejar hasta 30 conversaciones simultáneas a la vez. Aunque el formato digital de líneas E1 es diferente al formato T1, la central telefónica maneja todas las traducciones necesitadas del formato, de tal manera que los usuarios TAPI no necesitan hacer nada especial para manejar llamadas sobre líneas E1.

4.3.3 La red de servicios digital integrada (ISDN)

La Red de servicios digital integrada (ISDN) fue desarrollada para manejar voz, datos, y servicios de video sobre la misma línea. Aunque se desarrollo hace más de 20 años, las líneas ISDN recientemente se han puesto a disponibilidad en los principales mercados de servicio. La creciente expansión de servicios de datos (como la Internet), contribuye a aminorar el precio y aumentar la disponibilidad de ISDN en el futuro de los mercados.

La forma más común de servicio de la ISDN, llamada Interfaz de relación básica o BRI-ISDN, provee dos canales de 64Kbps y un canal de control de 16Kbps. La ventaja de ISDN es que los dos canales 64Kbps pueden ser configurados para proveer una sola tubería de 128Kbps para voz, video, datos, o simplemente para ser configurada para proveer canales de datos de 64Kbps y un canal digital de voz de 64Kbps simultáneamente. Así el servicio de la ISDN tiene previsto el acarreamiento de más de un tipo de medio al mismo tiempo. Además de los dos canales, BRI-ISDN provee un canal de control de 16Kbps, llamado canal D, que puede enviar señales de información y datos adicionales de control. Este control de datos puede contener información acerca del usuario que llama (el nombre, la localización, etcétera) u otra información suplementaria necesitaría para completar la transmisión.

4.3.4 El Pbx

El formato de Pbx es usado en los equipo comercial de conmutación para manejar sistemas telefónicos multilínea en oficinas. Típicamente, el proveedor telefónico local provee el servicio multilínea hasta el conector del PBX, para que entonces sean manejados todos los procesos de control y

conmutación de líneas dentro de la red del PBX. Los teléfonos del PBX operan en formatos propietarios y son usualmente líneas digitales.

El formato del PBX incluye normalmente más que simplemente la señal de voz / datos. Los impulsos para controlar luces relampagueantes en instrumentos telefónicos , datos que aparecen en paneles, e información adicional para manejar teleconferencia, son ejemplos de datos usualmente expedidos a lo largo de la misma línea.

El formato exacto de los datos de las líneas PBX depende directamente del vendedor-proveedores del servicios. Sin embargo, las aplicaciones TAPI pueden operar en todos los PBX, siempre y cuando el vendedor-proveedor este suministrando un servicios compatible con TAPI.

4.4 La API suplementaria de Telefonía para Dispositivos Telefónicos

La Telefonía Suplementaria también provee llamadas de función para el manejo de dispositivos telefónicos. Para TAPI, cualquier dispositivo que puede colocar o puede aceptar las llamadas, puede ser un dispositivo de teléfono. El conjunto de funciones telefónico de TAPI permite a los programadores inventar sus propios dispositivos telefónicos en código.

La Telefonía Suplementaria para dispositivos telefónicos puede estar divididas en los siguientes grupos de función:

- Las funciones básicas de manejo de teléfono proveen inicialización básica y cierre, abriendo y cerrando un dispositivo telefónico, y timbrando el dispositivo abierto.

- Configuración telefónica y las funciones de estado permiten a los programadores leer y escribir configuraciones diversas del dispositivo telefónico, como comportamiento de volumen, de ganancia, etcétera.
- Visor físico, datos, botón, y funciones de lámparas pueden usarse para leer y escribir información de visor para unidades de escritorio . Desde que TAPI puede soportar más que simplemente PC de escritorio, estas funciones permiten a un programa TAPI central a monitorear y actualizar pantallas LCD, hacer brillar intermitentemente lámparas, cambiar etiquetas de botones, y almacenar y rescatar datos de terminales .

La tabla VII presente en el anexo A de esta tesis muestra todas las funciones telefónicas Suplementarias de TAPI junto con breves descripciones de su uso.

4.4.1 Las Estructuras Suplementarias del Dispositivo del Teléfono de Telefonía

Las estructuras del dispositivo telefónicos más a menudo usadas son los PHONECAPS y estructuras PHONESTATUS. La tabla VIII presente en el anexo A de esta tesis muestra todas las estructuras telefónicas del dispositivo junto con descripciones breves de su uso.

4.4.2 Los Mensajes Suplementarios del Dispositivo del Teléfono de Telefonía

El dispositivo telefónico suplementario también usa una función de retorno (callback) para recibir mensajes de Windows. Esta dirección de recuperación es establecida durante la llamada de API del phoneInitialize.

Cada mensaje devuelve el mismo grupo de parámetros. La primera parte es el manejador del dispositivo telefónico. El segundo parámetro es el valor de recuperación (callback). Este valor siempre será el manejador de la corrida de la aplicación actual. Los siguientes tres valores disienten a merced del mensaje. Uno o más de estos valores de regreso contendrán datos de cero. La tabla IX presente en el anexo A de esta tesis contiene una lista de los mensajes Básicos de Telefonía, sus parámetros, y sus descripciones pequeñas.

4.5 Resumen

Es importante para saber que algunos tipos físicos de línea ofrecen opciones no disponibles en otros tipos de línea. Por ejemplo, TAPI no puede ofrecer los mismos servicios de línea de la ISDN que a otros tipos de líneas mas limitados. TAPI, sin embargo, provee una interfaz consistente para todos los servicios compartidos para todos los tipos de línea (como lineopen, dial, send data, close line, etcétera).

Como ya sabemos de capítulos anteriores, TAPI en su forma básica permite el manejo de las líneas telefónicas para proveer servicios básicos de telefonía, como llamar, contestar, Caller Id, etc. La Telefonía Suplementaria también provee llamadas de función para el manejo de dispositivos telefónicos, pero de una forma que aumenta el alcance de TAPI. Para TAPI, cualquier dispositivo que puede colocar o puede aceptar las llamadas, puede ser un dispositivo de teléfono. Sin embargo, mediante funciones extendidas, se pueden extender las funciones básicas de un dispositivo, de tal manera que el conjunto de funciones telefónico de TAPI permite a los programadores inventar sus propios dispositivos telefónicos en código, los cuales pueden funcionar incluso como un servidor telefónico.

Capítulo 5

Diseño de Aplicación telefónica

La aplicación de telefonía básica permite remplazar por completo el dispositivo telefónico de una sola línea con el uso de una estación de trabajo Windows, un módem de datos con soporte de librerías TAPI y una tarjeta de sonido adecuada con su correspondiente micrófono y parlantes. Para el desarrollo de esta aplicación se han considerado la funcionalidad actual de teléfono, ya que hoy en día no sólo debe permitir el intercambio de voz, sino que además ofrece servicios como el de máquina contestadora, mantener una llamada en espera y detección del número llamante.

En este programa se ha añadido código para manejar las situaciones típicas de un teléfono, incluyendo servicios extendidos disponibles en teléfonos especiales. Se debe manejar llamadas de entrada y con rumbo exterior, y manejar la información Caller Id de la llamada telefónica. Como agregado se dispone también de acceso a ventanas de diálogo para mantener un grupo de valores de configuración para el entorno del usuario. Se puede escribir un historial y notas de interés en cada una de las llamadas con rumbo exterior hecha a través de la aplicación y se dispone de un contestador automático. En este capitulo se explica cada uno de los procedimientos principales del programa, dentro de los cuales se realiza el proceso de ejecutar estas acciones.

5.1. Descripción

El objetivo propuesto para esta aplicación es la de tomar el puesto del dispositivo telefónico. Es por ello que primero se debe realizar un análisis de las acciones que ejecuta un teléfono normal, y otros con servicios para personas mas exigentes en sus necesidades.

Ante todo, el programa debe manejar perfectamente el proceso de realizar una llamada y de responder una llamada. Como complemento para las llamadas entrantes, cada día se vuelve mas común el uso del nuevo servicios de Caller Id en Ecuador, por lo cual es importante poder manejar esta trama de datos también dentro de la aplicación. Hay que recalcar que varios productos de Caller Id permiten programarse para que, en lugar de presentar únicamente el numero telefónico, presenten directamente el nombre del usuario que esta llamando.

Otros dispositivos telefónicos mas complejos disponen de contestadores automáticos para las situaciones en que no se alcanza a responder la llamada, no se quiere responder la llamada, o no se encuentran presentes para responder la llamada. Estos pueden almacenar, dependiendo de la situación, de media hora hasta una hora de mensajes telefónicos. De igual forma, algunos presentan la información sobre la llamada, como numero telefónico, fecha y hora de la llamada, en el momento de la reproducción de ellos.

Para finalizar, y sin estar directamente relacionado con el dispositivo telefónico en si, como complemento, las personas suelen tener una pequeña libreta telefónica o libreta de contactos con los números telefónicos de conocidos. Sin embargo, es común en nuestro medio que esta libreta

frecuentemente se pierda, se deteriore o que no se encuentre directamente a disposición en el momento de la llamada.

5.2 Entorno

Basándonos en la implementación de las bases de datos y su soporte para Caller Id y grabadora de mensajes, podemos encontrar usos prácticos tanto para uso doméstico u oficinas de empresas pequeñas y medianas, en donde el uso del dispositivo telefónico sea frecuente y en condiciones de ser optimizado por medio de un computador.

En el hogar para aquellas familias que mantienen un alto ritmo de uso del computador, pensado tanto para usuarios que gustan de llevar trabajo al hogar o de disfrutar de entretenimiento en el computador, el programa aprovecha el recurso telefónico no utilizado de la PC en ese momento. De esta manera, el usuario puede aprovechar las utilidades básicas del programa, como es llevar con facilidad una agenda telefónica y utilizar la identificación de llamadas por Caller Id.

En una oficina, la PC suele estar en funcionamiento la mayor parte del día útil. Así podemos prescindir del uso de un dispositivo telefónico al permitir que el computador maneje las llamadas e implementando bases de datos de la información. No solo tiene validez la opción de llevar una extensa base de datos de clientes y contactos telefónicos, sino que la opción de grabadora de mensajes permite hacer un mejor seguimiento de las llamadas entrantes, principalmente cuando el usuario ha tenido que retirarse momentáneamente de su lugar de trabajo.

5.3. Módulos principales de la aplicación

El manejo de las funciones telefónicas del programa se basan en la utilización de las funciones Tapi. Sin embargo, estas representan únicamente una parte de todo el flujo de procesos que se deben realizar para que el programa realice las acciones necesarias. La utilización de las funciones Tapi son solo el corazón de la aplicación telefónica, pero para llegar a su final utilización se debe realizar todo un proceso anterior que representa el verdadero trabajo de diseño e implementación de esta tesis.

Podemos separar el funcionamiento general del programa en cuatro grandes módulos, los cuales no son módulos independientes de ejecución lineal de instrucciones, sino que dependen en todo momento de los otros módulos del programa para su correcto procesamiento. Estas diferentes secciones del programa se las representa en la figura 5.1. El módulo base del programa es la que maneja el entorno del usuario, esto es, maneja los sucesos ejecutados por el usuario, y maneja los mensajes producidos por el sistema operativo. Mediante el análisis de esta información, se proseguirá llamando a las otras secciones del programa según sea necesario, de tal manera que producirá el resultado deseado por el usuario y se lo presentara en un entorno grafico adecuado.

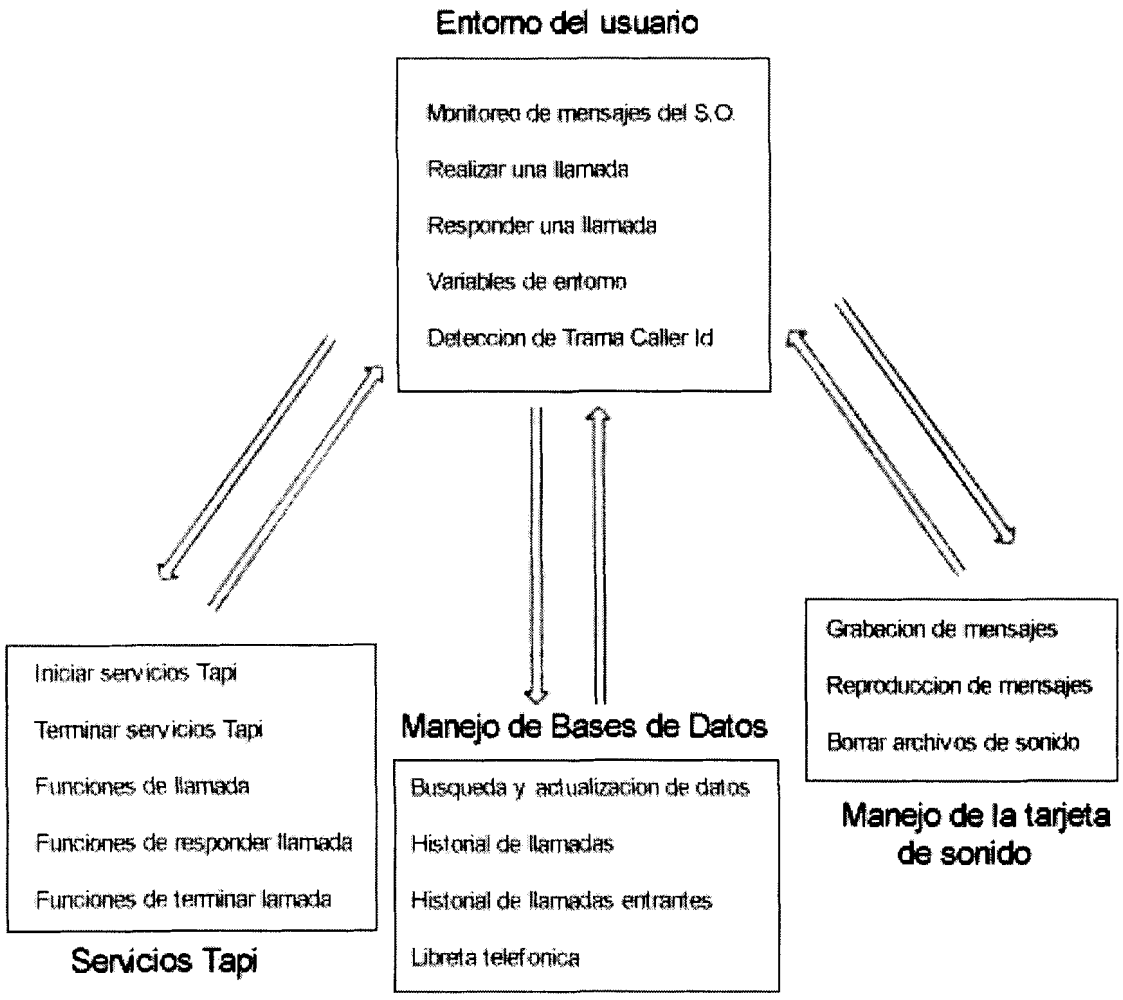

Figura 5.1 Módulos de la aplicación

Hay que recalcar que en todo momento es necesario mantener una comunicación entre estos diferentes módulos, lo cual se realiza tanto con variables globales de estado, como con llamados rápidos a procedimientos públicos en los otros módulos. Un claro ejemplo de esto es como después de realizarse un proceso independiente como el de llamar por teléfono o el de colgar, se llama al modulo de servicios Tapi.

5.3.1 Procedimientos

El manejo directo de los dispositivos telefónicos, se basaran casi directamente en las funciones Tapi, tanto para el manejo del modem o de la línea telefónica. Estas funciones serán llamadas como proceso final desde diferentes procedimientos, las cuales básicamente solo producirán mensajes de respuesta de "error - no error" desde sistema operativo, indicándonos la situación del proceso iniciado por la función Tapi utilizada. Le corresponderá al resto de la programación Visual Basic implementada con la función Tapi interpretar y procesar estas indicaciones del sistema operativo, según las necesidades de cada sección. Con este proceso se logra el entorno deseado de la aplicación

5.3.1.1 Proceso de inicio del programa

Cuando el programa se ejecuta, básicamente se debe manejar el inicio de los servicios Tapi, las conexiones con las bases de datos que utilizaremos a lo largo del programa, como son las del historiales y libreta telefónica, y preparar algunas variables de entorno básicas del programa, como las de indicación de estar reproduciendo y grabando mensajes de la grabadora, numero de mensajes, bandera de estar iniciados correctamente los servicios Tapi y las opciones de inicio del programa, como son minimizar y ventana inicial. Este proceso y su flujo se presentan en la figura 5.2.

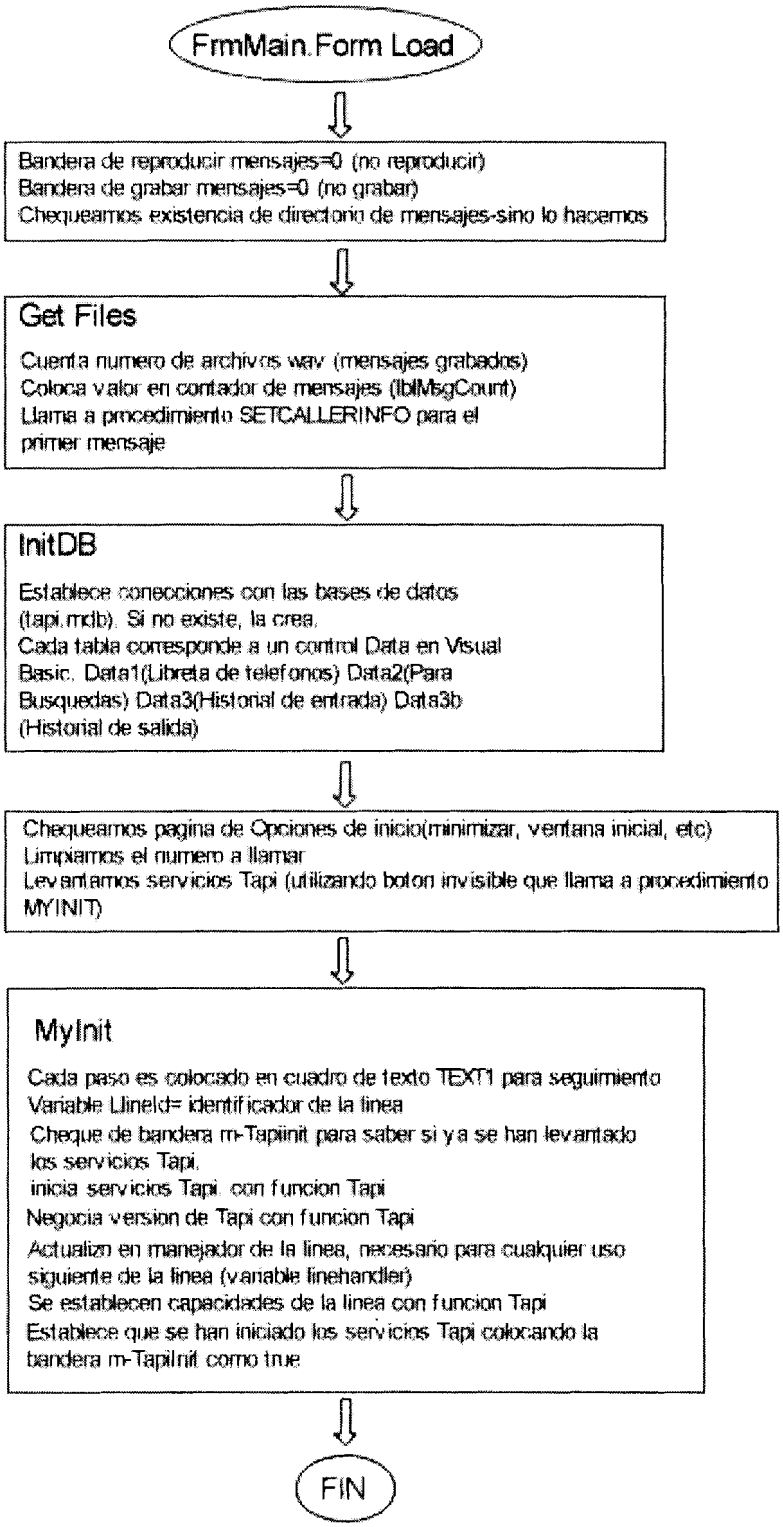

Figura 5.2 Pasos de inicio del programa

5.3.1.2 Proceso para realizar una llamada

Las siguientes secciones cubren los pasos del proceso de llamado con rumbo exterior. Cabe indicar que el paso final para este proceso reside en llamar a ciertas funciones Tapi, después de haber realizado todo un proceso introductorio. Estos llamados al servicio Tapi son expuestos en los pasos indicados por el flujo de la figura 5.3, el cual representa los pasos completos del proceso de realizar una llamada.

Básicamente el proceso introductorio a la utilización de las funciones Tapi reside en recabar los datos necesarios, como el numero telefónico en un formato correcto, determinar los datos completos de la persona a llamarse y realizar varios pasos redundantes para actualizar la libreta telefónica con estos datos. Finalmente se prepararan las variables globales que indicaran que se esta realizando una llamada y que no pueden realizarse otros procedimientos, como por ejemplo el de reproducción de la grabadora de mensajes, y se determinan los parámetros que necesitan las funciones Tapi para realizar la llamada.

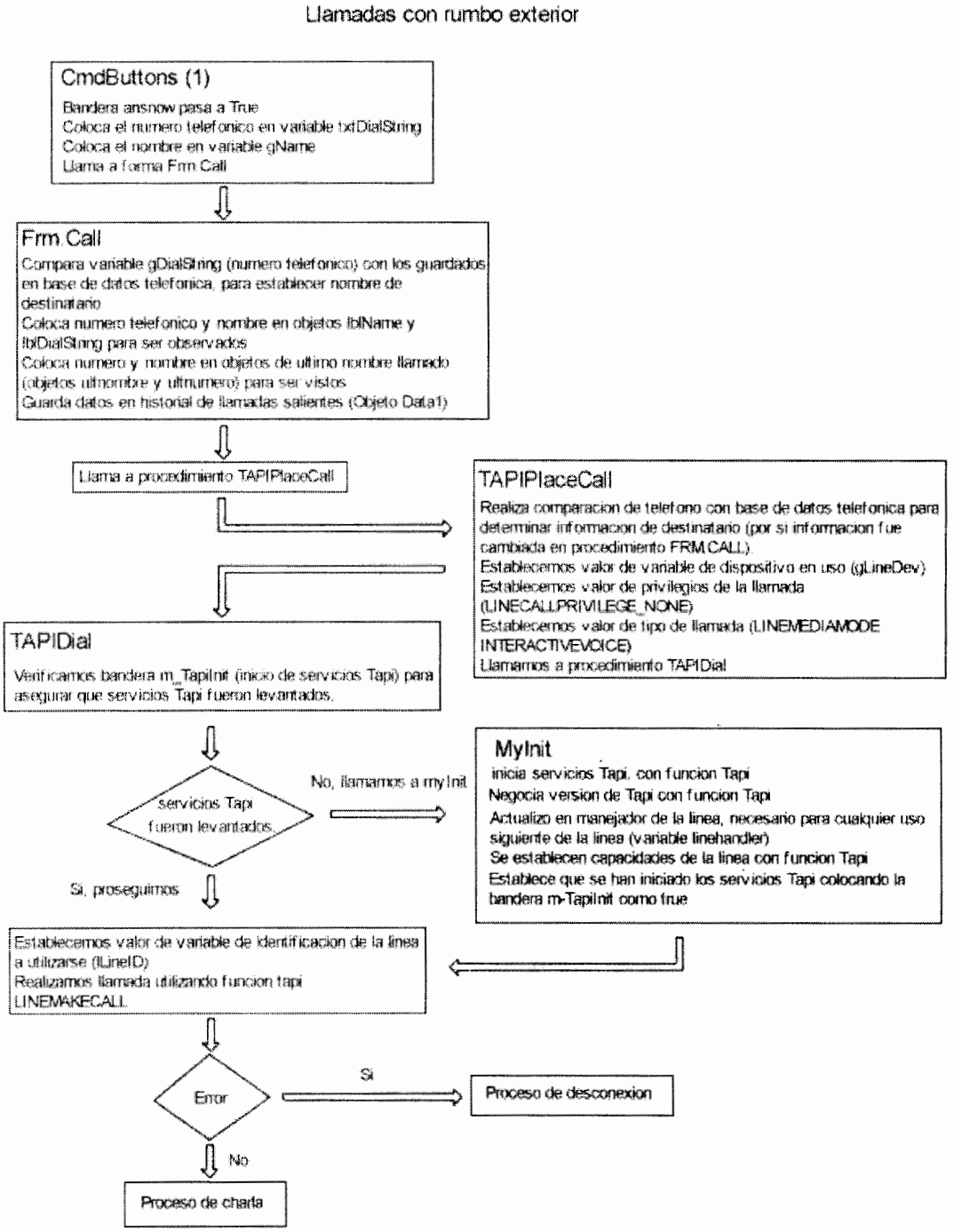

Figura 5.3 Pasos para llamadas con rumbo exterior

Como podemos ver, dentro del proceso de llamada se utilizaron algunas funciones Tapi, las cuales serán explicadas a continuación. Hay que recalcar que el orden de utilización de estas funciones si es importante .

- **lineInitialize** llama a poner en marcha la sesión TAPI. Después de la inicialización exitosa, la rutina devuelve un valor en el parámetro del lineHandle. Se usará este valor a todo lo largo de la sesión TAPI. También se definirá una cuenta del número total de líneas TAPI para este puesto de trabajo. Se usará la información para comprobar la versión de TAPI y los parámetros de línea de cada una de las líneas antes de se intente colocar una llamada.

- **lineNegotiateAPIVersion** llama a asegurar que se puede usar los servicios TAPI instalados. Se necesita comprobar la capacidad de cada línea disponible porque es posible que se demande una versión de TAPI que no este disponible para esta máquina.

- **lineOpen** llama a obtener una línea que sea apropiada para las necesidades (los datos, el fax, voz, etcétera). Se necesita chequear cada línea y demandar el nivel apropiado de servicio. Por ejemplo, si se quiso colocar una llamada de voz interactiva, entonces se usaría la función del lineOpen para localizar una línea que soporte voz interactiva. Éste es un punto importante. Esta dentro de lo posible que el puesto de trabajo defina varios dispositivos TAPI, pero puede haber solo uno que puede proveer el tipo de servicio que se necesita (voz, el fax, los datos, etcétera). Si no hay dispositivos disponibles (ninguno existe o el actual esta ocupado), entonces se obtendrá un mensaje de error. Sin embargo, si una línea apropiada esta disponible, entonces se recibirá un cero como un código de regreso y un valor indicando el

manejador de línea abierto. Se usará este valor en subsiguientes llamadas TAPI.

- Usar la estructura **LINECALLPARAMS** para establecer los parámetros antes de que se de lugar a la llamada. Se usa esta estructura para decir a TAPI que determine la velocidad y el soporte lógico informático de su llamada (datos, voz, etcétera) y otros valores. Determinar la estructura LINECALLPARAMS es optativo. Si se no determina cualquier apreciación especial para los LINECALLPARAMS, entonces Microsoft TAPI usará los valores predeterminados. Para la mayoría de llamadas, los valores predeterminados surtirán efecto sin ninguna complicación.

- **lineMakeCall** llama a realmente intentar colocar una llamada TAPI. Esta función pasa la variable que contiene el número de teléfono para llamar, una agarradera para la línea abierta (que se obtuvo de lineOpen). Es importante anotar que en este punto la llamada, esta solamente se ha colocado, pero no se ha completado. Todo lo que TAPI sabe por seguro es que los dígitos han sido marcados y que la línea telefónica esta activa.

Durante el proceso de completar estos pasos, se utilizan los mensajes admitidos por la función de recuperación de mensajes para rastrear el progreso de la llamada y responder consecuentemente. Corresponde al modulo de monitoreo del mensajes del sistema operativo realizar el seguimiento de la llamada en el sentido si se la realizo con éxito o no.

5.3.1.3. Proceso para las Llamadas Entrantes

Hay un total de seis pasos básicos para usar a TAPI para procesar llamadas de entrada (Figura 5.4). El proceso inicial para las llamadas entrantes es similar al proceso para realizar una llamada.

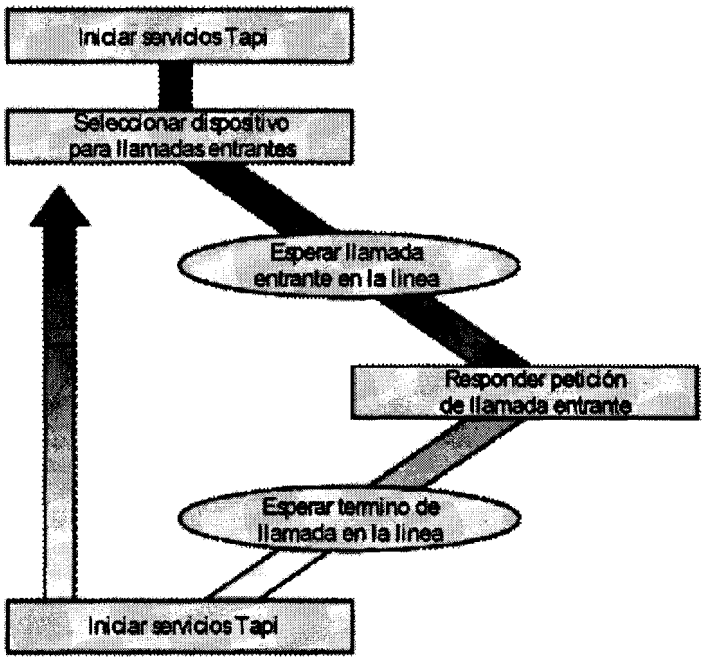

Figura 5.4 Pasos de procesamiento de llamadas

Los seis pasos básicos son:

- Iniciar servicios TAPI en el puesto de trabajo.
- Chequear dispositivos selectos, abiertos y válidos de línea para las llamadas entrantes.
- Esperar una llamada de entrada.
- Cuando una llamada aparece, aceptar la llamada y comenzar conversación.

- Cuando la llamada pone fin, cerrar la línea y prepararse para la siguiente llamada.
- Cuando la sesión esté terminada, cerrar servicios TAPI en el puesto de trabajo.

El proceso real implementado resulto ser algo mas complejo. Esto se debe principalmente al hecho de que en este caso se debe realizar mas acciones agregadas al proceso normal de contestar, como por ejemplo realizar la comprobación y obtención de los datos de Caller Id y alternar el manejo de la tarjeta de sonido del computador con el proceso de responder al llamado.

Como el programa es diseñado para esperar una llamada, es el teléfono el que realmente controla el flujo inicial del programa. El flujo de procesos implementados en el proyecto se lo explica con la figura 5.5.

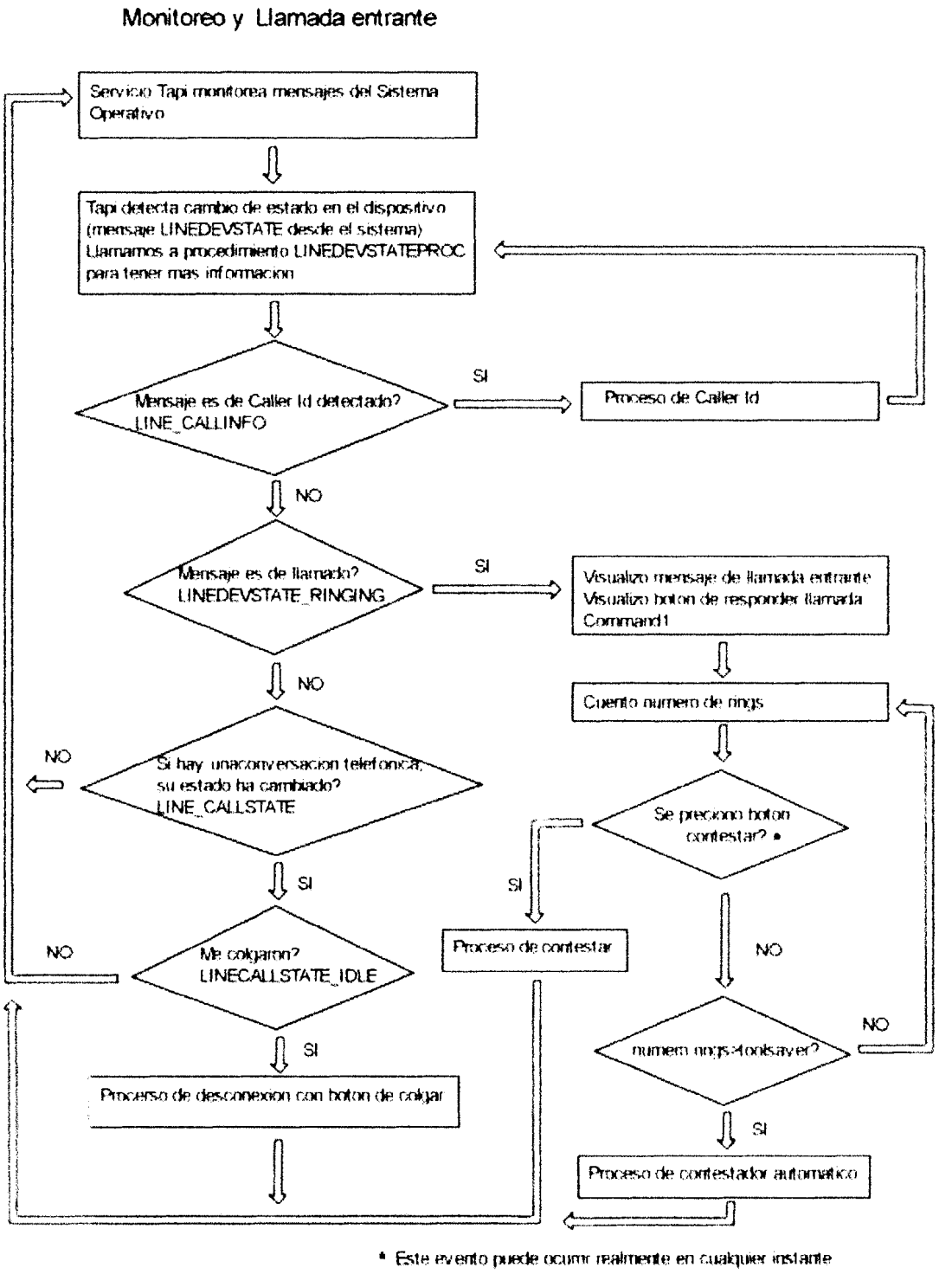

Figura 5.5 Flujo de procesamiento de llamadas

Primero, hay que recalcar que el proceso de responder una llamada puede efectuarse de dos maneras totalmente diferentes, pero el inicio para ambas es el mismo. Se debe esperar un mensaje del sistema operativo indicando que se ha efectuado un ring en el dispositivo telefónico, lo cual indica una llamada entrante. Primero aparecerá un botón de responder, el cual antes se encontraba invisible para el usuario. Al mismo tiempo que esto se realiza, se inicia un contador de rings. Precisamente basándonos en estas dos acciones es que se dispone de dos formas de responder la llamada.

Las dos formas corresponden a presionar el botón de contestar, o el proceso del contestador automático. En el caso del primero, el presionar el botón iniciara el proceso de responder implementado con las funciones Tapi. En el caso del segundo, primero se realizara una comprobación del numero de timbradas. Una vez que este sea igual a la variable Tollsaver, se llamara al mismo proceso de contestar implementado con Tapi. Sin embargo, cada proceso proporciona valores diferentes de la variable global que inicia el proceso de reproducción y grabado de la contestadora. Es así que en el primer caso, la respuesta al llamado deberá ser una conversación tipo "Voz interactiva", y en el caso del segundo, no se habilita al micrófono para responder, sino que se le asigna privilegio de tarjeta de sonido a la reproducción wav (que reproduce el mensaje de saludo), para pasar finalmente al modo de grabación, que terminara creando un mensaje mas en la grabadora de mensajes. Este proceso termina con la detección del los tonos de colgado, con lo cual se llama a el proceso de desconexión, explicado mas adelante.

Al mismo tiempo que se inicia el monitoreo de las timbradas, se realiza una comprobación de la trama de datos del timbrado, para comprobar la existencia de información tipo Caller Id. Este proceso, al ser separado del

proceso normal de responder una llamada, se lo analizara mas delante de forma exclusiva.

El proceso principal reside en la interpretación de los mensajes del sistema operativo y determinar los datos involucrados en esas tramas de datos. Cuando TAPI indique que una llamada aparece en la línea seleccionada, el programa recibirá un mensaje con el manejador para la llamada y la respuesta. Estas serán utilizadas por el método de líneanswer. Una vez esto esta terminado, el programa puede pasar la llamada al usuario y espera hasta que la conversación es completada.

5.3.1.4 Proceso de desconexión

Hay básicamente dos formas para que una conversación telefónica se termine: El usuario local cuelga el teléfono o la persona en el otro extremo cuelga el teléfono. En el primer caso, se puede usar el lineDrop y métodos lineClose para terminar la llamada. Sin embargo, si el usuario remoto cuelga el teléfono, la línea es cerrada automáticamente por TAPI.

Además se necesita liberar en el programa al manejador de la llamada (usando a lineDeallocateCall). En ambos casos, el lineClose o lineDeallocateCall ya prepararán al dispositivo de línea para recibir nuevas llamadas.

Basándose precisamente en el proceso de desconexión requerido en el programa, se decidió implementar un proceso único de desconexión el cual es llamado desde cualquier parte del programa que requiera terminar los servicios Tapi. Lo que se realiza esta dentro del proceso de hacer clic en el botón colgar, sea esto realizado por el usuario para terminar la llamada, o el

programa realiza un clic virtual del botón cuando detecta que han colgado desde el otro lado de la línea. En este flujo se implementaron las llamadas a las funciones explicadas arriba de desconexión. El flujo esta representado por la figura 5.6.

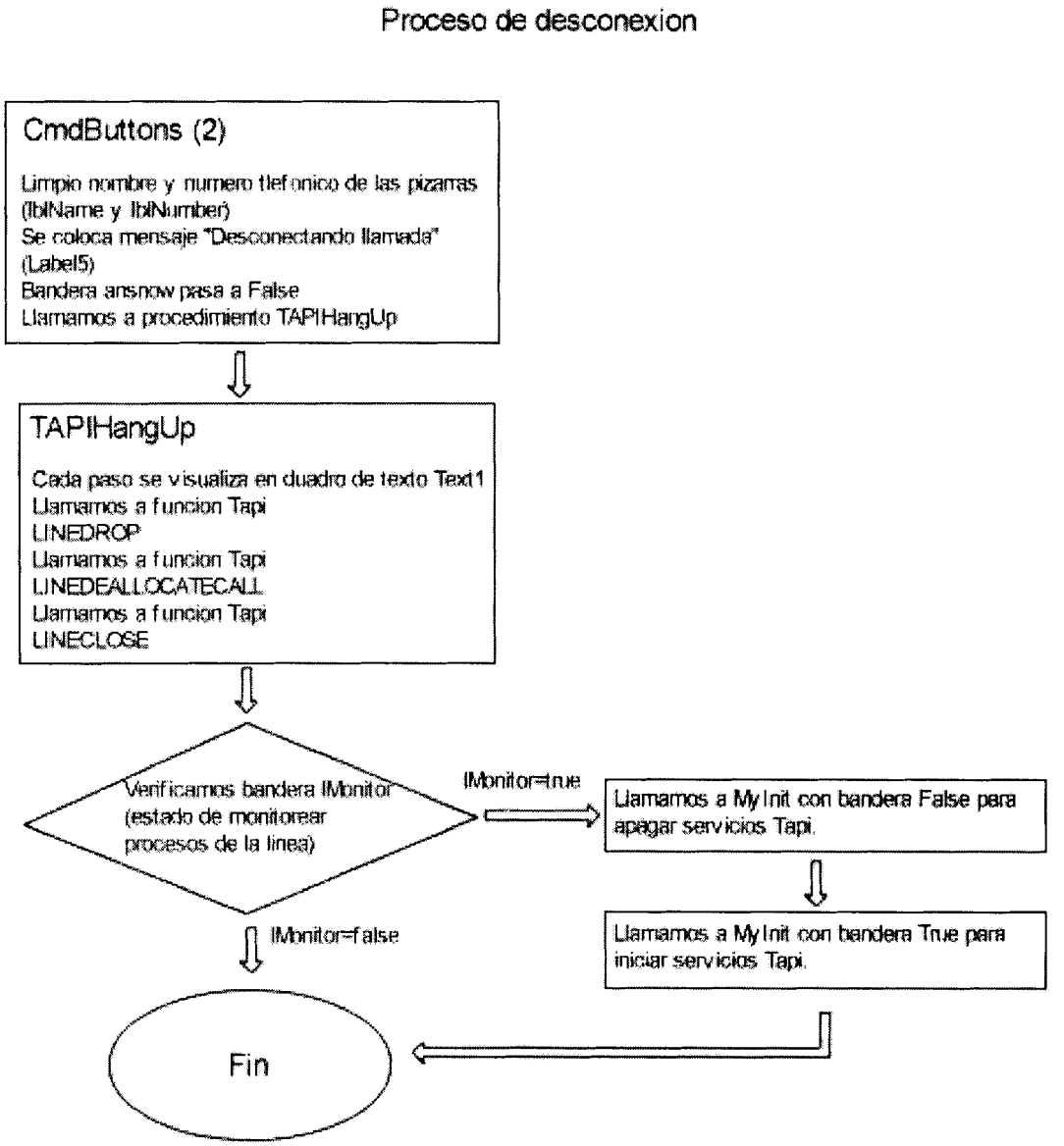

Figura 5.6 Proceso de desconexión

Hay dos puntos centrales para recordar al desarrollar aplicaciones que acepten llamadas de entrada. Primero, después de abrir el dispositivo usando los parámetros que informan a TAPI que se desea ser notificado de cualquier llamada de entrada, se necesita esperar y chequear los mensajes TAPI que ocurren cuando una llamada aparece. El segundo punto central es que, una vez que una llamada es aceptada y en marcha, el programa necesita esperar mensajes TAPI que indiquen que la llamada ha acabado. En otras palabras, su manipulador de entrada de llamada tiene dos trabajos principales:

- Espere una llamada entrante.
- Espere una llamada por acabar.

5.3.2 Otros módulos del Programa

Proseguiremos analizando los módulos que realizan el trabajo de procesar la información especial en el programa. Básicamente son procedimientos que no involucran funciones Tapi, sino que por el contrario funcionaran analizando la información que le será proporcionada por Tapi en los diferentes estados de la llamada, para ser analizados y manipulados según las necesidades. Otros procedimientos son totalmente independientes de las funciones Tapi y sirven para mantener y producir el entorno del usuario.

5.3.2.1 Procedimiento de Inicialización de Servicios Tapi

La función de inicialización de servicios TAPI recibe el nombre de Myinit, y es la función que es llamada a lo largo de todo el programa para lidiar con el inicio y termino de los servicios Tapi. Sin embargo, el nivel de inicialización de los servicios TAPI depende de la situación en que es llamada esta función. Es así que se decidió manejar este proceso en un solo procedimiento, el cual puede ser llamado desde cualquier parte del programa, pero implementando una bandera de indicación para iniciar o apagar los servicios Tapi. De igual manera, el procedimiento MyInit también involucra manipular varias variables de entorno del programa y variaciones en la presentación física del programa. El flujo de funcionamiento de este procedimiento se lo muestra en la figura 5.7.

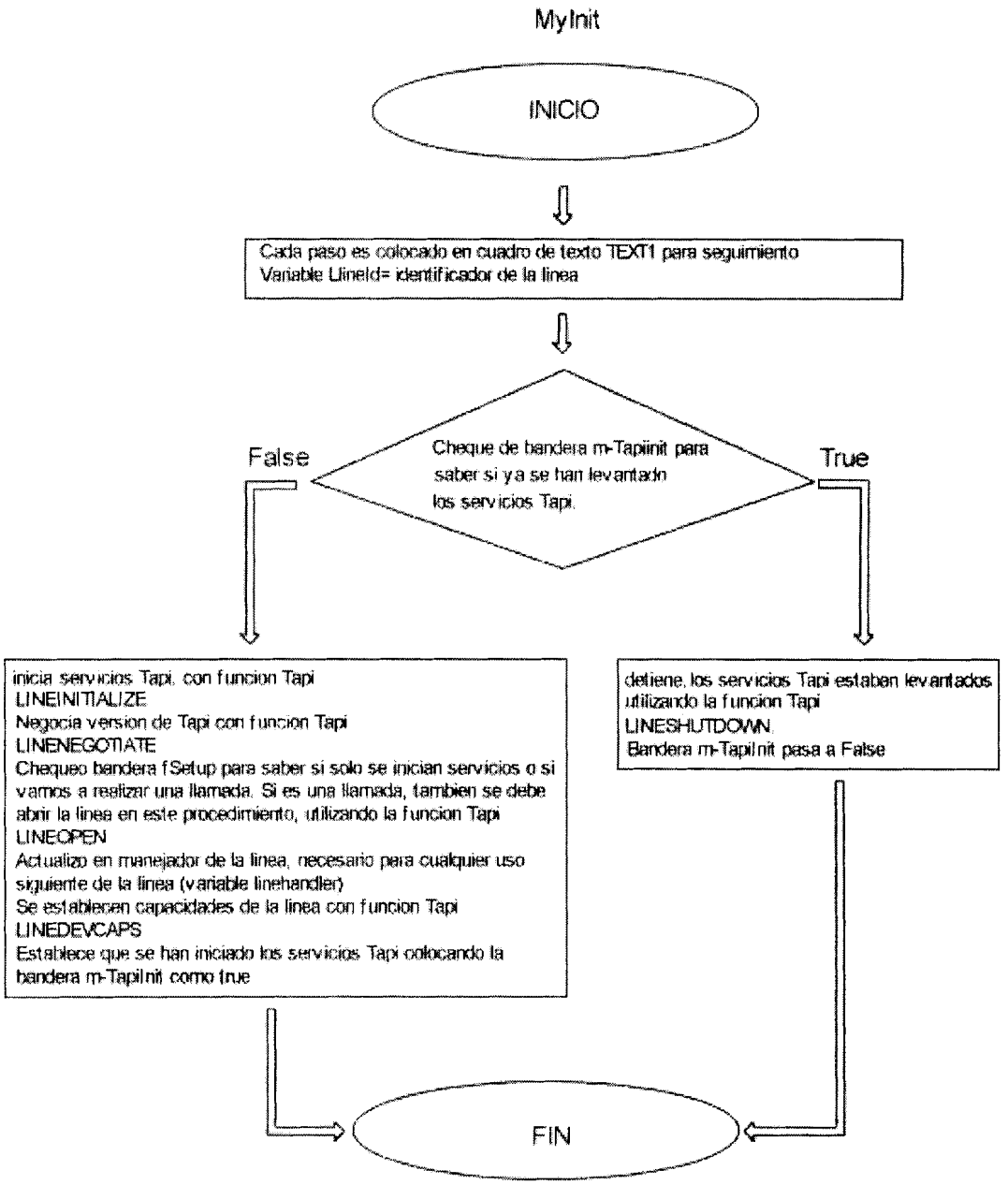

Figura 5.7 Flujo de procesamiento MyInit

Esta función puede ser llamada básicamente en dos situaciones. La primera es desde la ventana de configuración de los dispositivos, esto es, dlgsetup. Este llama a myInit tomando además la bandera fsetup para indicarle que es un inicio temporal únicamente para configuración. Es así que

en este caso, solamente se inicializa los servicios TAPI, y se negocia una versión. Si la inicialización no es para configuración, esto significa que es para el funcionamiento normal de la aplicación. Es así que en este caso, además de los procesos ya mencionados, se abre la línea y se configura la aplicación para recibir todos los mensajes de respuesta de TAPI. Además en esta función contamos con la bandera m_TAPIInit, la cual se utiliza para indicar si TAPI ya esta iniciado o no. Si esta inactiva, esta función iniciara a TAPI según ya se ha indicado. Casi contrario, esta función se encargara de cerrar todos los servicios TAPI, dejando finalmente la bandera de inicio m_TAPIInit en falso.

5.3.2.2 Ventana de configuración

Los usuarios pueden determinar varios parámetros para controlar el comportamiento de la aplicación, directamente desde esta sección del programa. Estos valores de control son almacenados en el registro de Windows y cargados cada vez que el programa es llamado. Los valores son almacenados en el HKEY_CURRENT_USER\Software\Visual Basic. Los valores cruciales almacenados allí son:

- Minimizar al iniciar: Cuando se le asigna el valor de "1", al cargarse la aplicación, esta inmediatamente se minimiza a sí misma. Los usuarios pueden hacer clic en la aplicación colocada en la barra de inicio siempre que deseen colocar llamadas salientes. Cuando una llamada entra, la hoja de solicitud automáticamente aparece maximizada en el puesto de trabajo del usuario. El valor preestablecido es "0""
- Llevar historial: Esto es determinado con el valor de "1". Es entonces cuando todas las llamadas con rumbo exterior son almacenadas en el historial de mensajes de salida. El valor predefinido es "1"

- Número de origen: Contiene el número de teléfono desde el cual las llamadas se originan. El valor preestablecido está en blanco.
- Pagina de inicio: Determina que con el valor de "1", la página de la Guía telefónica aparece en el arranque. Cuando esto sea determinado con "0", la página del Dial aparece al inicio. El valor establecido es "0".

además se ha dispuesto otra ventana de configuración solo para el manejo del MODEM. En esta ventana de configuración se ha implementado principalmente lo que se refiere a el dispositivo y al Tollsaver. La sección de enumeración de los dispositivos TAPI funciona en conjunto con la función myInit, que es la de inicio de servicios TAPI. Realmente es esta función la que inicia los dispositivos, para simplemente ser recogidos por esta ventana para su exposición. Ésta es una solución buena desde que la enumeración de los dispositivos TAPI es larga y de esta forma se esta usando una sola vez el proceso di búsqueda de ellas, para ser inmediatamente guardada en el registro de Windows para su posterior uso, incluyendo en el proceso de correr la aplicación. Como ya hemos explicado antes, frmMain.myInit es llamado con una bandera denotando su uso para esta sola forma. así, todo lo que la función hará es iniciar TAPI y tratar de negociar una versión mínima de TAPI 1.4. Una vez que hemos completado inicialización, circulamos a través de todos los dispositivos detectados y los metemos con sus nombres en el cuadro de lista cmbDevice. En esta lista un dispositivo debe ser seleccionado, y ese el dispositivo que usaremos para toda la aplicación.

El ITollSaver sirve para recuperación remota de mensajes. Si el código de acceso correcto es introducido se llama a la función de sonido. Si se pasa mas de un segundo sin un dígito introducido, entonces este es puesto a cero dar otra oportunidad.

5.3.2.3 Procedimiento de recuperación de mensajes de Windows

La función de recuperación de mensajes de Windows se la denomina con LineCallBack, y es el corazón del manejo de TAPI. Realizar un procedimiento de lectura de mensajes de respuesta del sistema operativo es básico, ya que dependiendo de estos mensajes es que podemos conocer el estado de la comunicación y proseguir con su manejo. Esto es básicamente la situación expuesta en la figura 5.5. TAPI envía todo estado de llamada a la aplicación a través de esta función de recuperación. Si el dwParam2 es diferente a cero, significa que un error ha ocurrido. Usamos una pequeña subrutina de manejo de errores al final de la función para manejar estos posibles errores del dwParam2 menor a cero. Cualquier cosa en la que estamos interesados o deseemos manipular, obtener más información, o actuar depende de los sucesos presentados por esta función, Algunos de los sucesos que principalmente nos interesan del CallBack son:

- Todos los cambios de estado de la línea, los cuales serán analizados después por una función separada llamada a partir de aquí
- Todos los cambios de estado de la llamada, los cuales serán analizados después por una función separada llamada a partir de aquí
- Generar un tono después de reproducir nuestro saludo. Este mensaje está generado por TAPI para indicarnos que es momento de grabar y registrar un mensaje (opción de máquina de grabar mensajes).
- Indicar si se ha traído alguna información de Caller Id, lo cual puede ser posteriormente procesado mediante el uso de GetCallerInfo.
- Mensaje de lineMonitorDigits indicándonos cuando se ha detectado un tono, cuando realizamos el proceso de responder a la llamada.

La naturaleza de la respuesta de la función callback también puede ser examinada basándose en algunos parámetros de su respuesta. Principalmente chequearemos el parámetro dxParam2. Si el dwParam2 difiere de cero significara para nosotros que un error ha ocurrido. Con este parámetro será también como manejaremos la rutina de errores.

5.3.2.4 Procedimiento para obtener trama de Caller ID

GetCallerInfo es la función encargada de realizar el trabajo de Caller Id. Básicamente lo que hace es tomar un manejador de la llamada para proseguir realizando una petición a la función TAPI lineGetCallInfo, la cual, de forma básica, debería ser capaz de extraer la información de Caller Id. Sin embargo, esta función no es infalible y puede fallar en algunas situaciones de llamadas. Es por ello que se implementa un proceso de chequeado del paquete (realizado con funciones Visual Basic) para indicar si fue correctamente procesado. El flujo del procedimiento se indica en la figura 5.8.

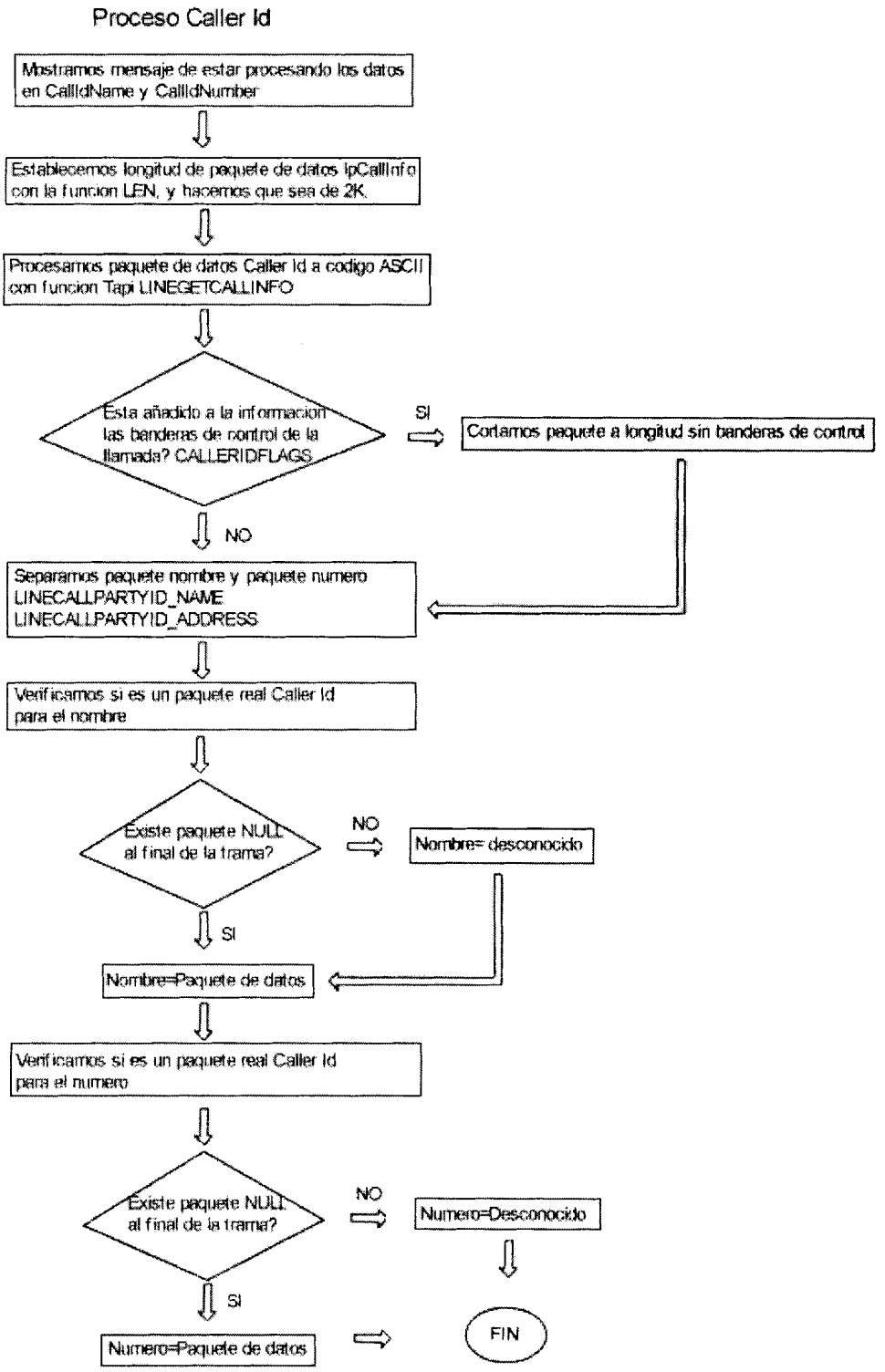

Figura 5.8 Flujo de procesamiento del Caller Id

Básicamente el problema reside en la longitud del mensaje Caller Id. Hay que suponer que éste es una estructura de largo variable. Sin embargo, el control de estructuras variables no es la especialidad de Visual Basic, por lo que el manejo de este campo con funciones TAPI llamadas desde VB no es fácil, y puede fallar. La solución parcial puede ser el definir una longitud básica fija del mensaje. Normalmente la longitud del mensaje puede ir desde 1 k hasta 640 K en algunas situaciones, sin embargo, el estándar comúnmente usado es de 2k. Es así que se ha añadido un campo bByte () hasta el fin de la estructura, y se lo ha colocado en 2k. Hay más formas complicadas para hacer esto de forma más dinámica. CopyMemory podría ser usado, considerando el hecho de que lo utilizado es realmente un truco. Finalmente realizamos este artificio con todas las constantes del LINECALLPARTYID _ para ver si la información del número/nombre(si lo hay) es OUTOFAREA, BLOQUEADA, etc. En lugar de eso justamente se coloca "desconocido" en las variables globales CallIDNumber y CallIDName.

5.3.2.5 Determinar parámetros de los mensajes del sistema operativo

El procedimiento LineDevStateProc es el encargado de realizar la tarea de procesar en mayor detalle las variables recibidas desde el sistema operativo. Este modulo es llamado en la línea "LÍNE _ LINEDEVSTATE", recibida desde los mensajes de LineCallBack, durante el proceso de monitorear las llamadas entrantes. Precisamente su función será la de estudiar en mayor profundidad las variables recibidas en este estado acompañando al mensaje LÍNE _ LINEDEVSTATE . Esto proveerá mas información de forma mas concisa, con lo que se podrá usar para determinar si hay que responder mensajes, con que características esta el llamado, etc. En cuántas timbradas respondemos depende directamente de si tenemos

habilitado el contestador para cualquier mensaje o no, es decir, es función típica de TollSaver configurado.

5.3.2.6 Determinar parámetros de estado de la llamada

El procedimiento LineCallStateProc es el que se encarga de manejar exclusivamente el análisis de los mensajes del sistema operativo concernientes al estado de la lamada. Este modulo es llamado en la línea "LINE _ CALLSTATE", recibida desde los mensajes de LineCallBack, de forma similar al procedimiento anterior, durante el proceso de monitoreo de llamadas entrantes. Precisamente su función será la de estudiar en mayor profundidad las variables recibidas en este estado acompañando al mensaje LINE _ CALLSTATE. Aquí se determinara las características de CallerID en caso de que no existe un mensaje del tipo LINE _ CALLINFO, el cual usábamos para la función de Caller Id al indicarnos que existe información de la llamada. Otras funciones importantes ejecutadas aquí son las de responder a la llamada como resultado del mensaje LINEDEVSTATE _ RINGING en el procedimiento LineDevStateProc, por lo que aquí se termina de definir que estamos conectados. Otras funciones son las de descartar (Drop) la llamada en caso de que nos cuelgan o de que la llamada fue descartada en nosotros mismos. En este caso, simplemente se pone a cero algunas variables y nos prepara para esperar a la siguiente llamada.

5.3.2.7 Manejo de Bases de Datos

Hemos desarrollado 2 subrutinas principalmente para lo que es el manejo de las bases de datos. Esta base de datos se denomina TAPI.mdb, y se compone de 3 tablas, TAPI, Log y Log2. Estas tablas son las encargadas

de manejar los aspectos de la libreta telefónica y del historial de llamadas, siendo justamente la tabla TAPI la destinada a la libreta telefónica, y las tablas del tipo Log las utilizadas para los historiales (los historiales de las llamadas entrantes y con rumbo exterior están separadas). Cada una de estas tablas tienen habilitadas su bandera de escritura, por lo que podrán ser manejadas y cambiadas dentro del programa.

La tabla TAPI se compone de los campos Nombre,del tipo TEXT(30), teléfono del tipo TEXT(20), Llamado del tipo DATE. Las tablas Log se componen de los campos Nombre del tipo TEXT(30), Llamado del tipo DATE, teléfono del tipo TEXT(20), Comentario del tipo MEMO.

Primero, disponemos de una rutina llamada InitDB, que establece las conexiones iniciales con nuestra base de datos, y determina sus características. Si el intento de establecer conexión falla, se debe tratar de la inexistencia de la base de datos, por lo que hemos implementado también la rutina MakeDB, la cual será la encargada de crear esta base de datos.

5.4 Resumen

Básicamente debemos separar al proyecto en 4 secciones principales, las cuales deberán manejar los aspectos de telefonía, Caller Id y bases de datos.

La primera sección deberá encargarse de inicializar los servicios TAPI y fundamentalmente, manejar el proceso de monitorear llamadas, componiéndose esta sección de varios procedimientos, incluyendo el de monitoreo de mensajes de comunicación con Windows (Callback). Partiendo

de este procedimiento, se desprenden otros procedimientos de importancia variable.

La siguiente sección seria la de manejo de las llamadas salientes, incluyendo colocar la llamada y desconectarla dependiendo de el estado de ella.

La sección Caller Id se desprende directamente de la sección de monitoreo, pero se la considera separada porque su manejo es especializado. Este procedimiento es funcional, pero no universal en el sentido de que su confiabilidad depende mucho tanto de las propiedades del MODEM como de las características de la línea y servicio Caller Id.

Finalmente, el manejo de las bases de datos se toma como sección separada. Estas bases de datos son manejadas tanto para la libreta telefónica como para el historial de llamadas, y se compone principalmente de su inicialización, creación y actualización.

Capitulo 6

Pruebas del funcionamiento del programa

Para constatar el funcionamiento del programa, se realizaron varias pruebas en diferentes hogares de la ciudad de Guayaquil . Para ello fue necesario preparar algunos requerimientos para el programa. Los principales fueron:

- Contratar el servicio Caller Id en la central telefónica local, en este caso, Pacifictel
- Constatar el uso de un computador con el hardware mas apropiado para las pruebas. En este caso, un MODEM del tipo MODEM de voz.

Las pruebas se compusieron de verificar la identificación de la trama Caller Id en tres diferentes tipos de llamada: llamada local, llamada de celular y llamada desde otra provincia.

6.1 Prueba de llamada local

Esta prueba fue realizada desde un hogar en la ciudadela Alborada 7 Etapa, localización que dispone de servicio de Caller Id desde aproximadamente 2 meses hacia la fecha actual. La maquina a utilizarse fue una Pentium II, 350 Mhz y 32 Mb de memoria, utilizando Windows 98. La figura 6.1 nos muestra la configuración del MODEM a utilizarse. Es un V.90 PCI MODEM configurado para responder automáticamente con la grabadora de mensajes al quinto timbrado, grabar el mensajes con una longitud máxima de 60 segundos, o permitir el acceso remoto a los mensajes con una contraseña (123) al tercer timbrado.

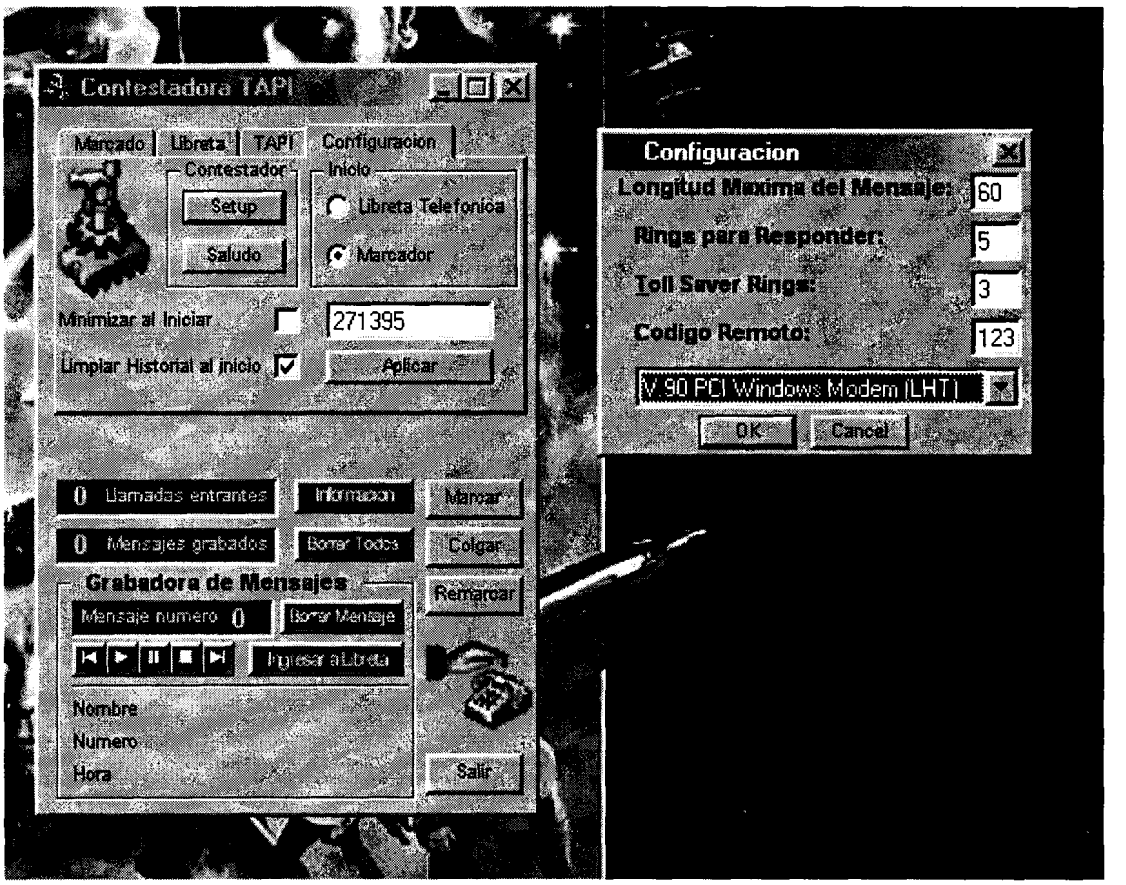

Figura 6.1 Configuración del programa

Además, en la figura 6.2 podemos apreciar las condiciones iniciales del programa. Los historiales han sido limpiados, por lo que no hay registro de llamadas entrantes ni con rumbo exterior. Los contadores de mensajes entrantes y de la maquina contestadora están en cero. De forma similar, no hay información de Caller Id. Finalmente, podemos apreciar como el proceso de inicialización de los servicios Tapi han sido realizados satisfactoriamente.

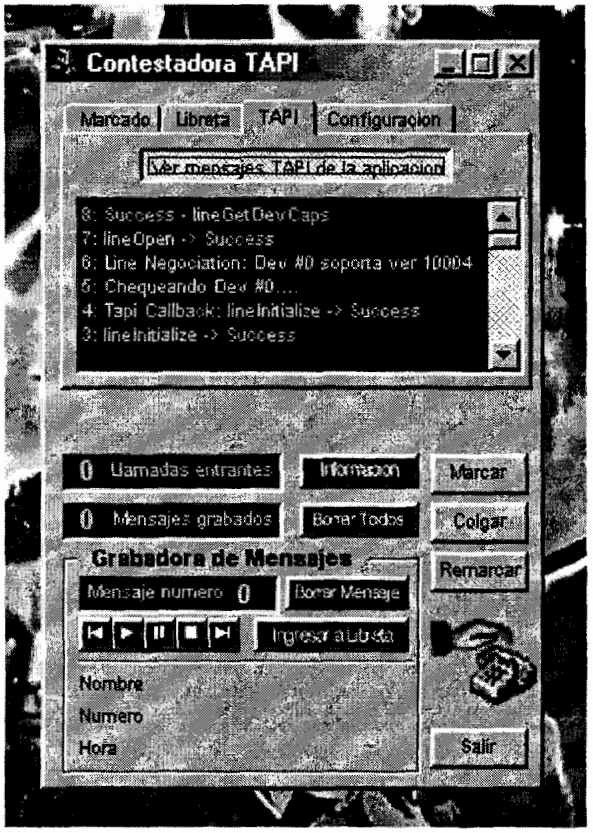

Figura 6.2 Condiciones iniciales del programa

La figura 6.3 nos muestra el proceso de identificación de una llamada entrante. Ante todo hay que explicar que esta imagen fue tomada

aproximadamente al tercer timbrado. La razón se debe a que no todos los campos de identificación de la llamada son mostrados al mismo tiempo. El primer campo a ser mostrado es el de el horario. A partir del segundo timbrado será mostrado el campo de teléfono, al haberse finalizado el proceso de enviado de la trama Caller Id en el segundo tomo de timbrado. Finalmente el campo nombre es el ultimo en ser mostrado, debido a que en la trama Caller Id local no viene incluido el nombre, por lo que el programa ejecutara una búsqueda en su base de datos interna para intentar llenar este campo. En este caso, si encontró un nombre asociado al teléfono. Hay que recalcar que aparece el numero 1 en el campo de llamadas entrantes, mientras que el el campo mensajes grabados aun esta en cero, debido a que no se ha grabado mensaje alguno aun. Además, se ha habilitado el botón Contestar, abriendo la posibilidad de responder el llamado telefónico mediante un micrófono y audifonos.

Figura 6.3 Llamada entrante

Finalmente tenemos el proceso de desconexión de la llamada, mostrado en la figura 6.4. Para este caso, no hemos respondido al teléfono y hemos permitido al programa grabar el mensaje. Este queda almacenado en el directorio de mensajes y se actualizara el contador de mensajes grabados. Hay que recalcar el hecho de que la desconexión fue automática desde el hecho de que el usuario que llamo colgó el auricular, lo cual será detectado por el programa e iniciara el proceso de cerrado de la comunicación. El botón contestar desaparecerá y será nuevamente reemplazado por el botón marcar.

Figura 6.4 Desconexión de la llamada

Finalmente podemos apreciar en la figura 6.5 como la información detectada por el programa corresponde perfectamente a la información mostrada por un típico dispositivo comercial de Caller Id usado ampliamente en la ciudad.

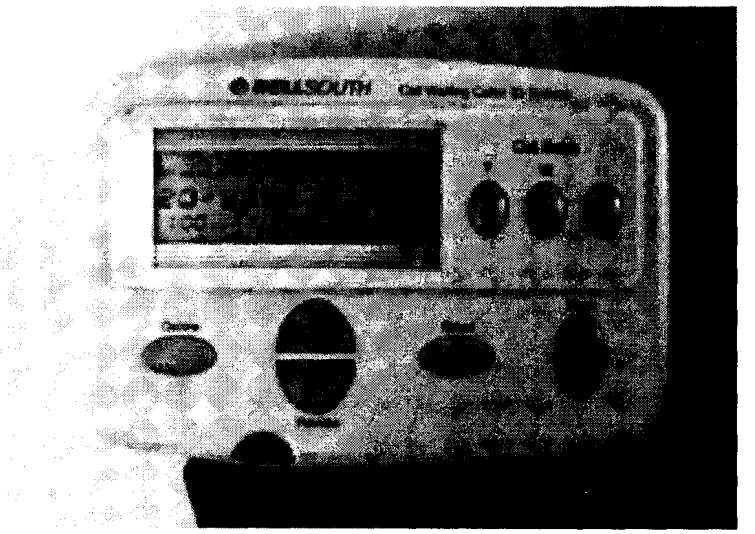

Figura 6.5 Chequeo de Información Caller Id

6.2 Prueba de llamada desde un celular

Esta prueba fue realizada desde un hogar en la ciudadela El Paraíso, localización que dispone de servicio de Caller Id desde aproximadamente 2 meses hacia la fecha actual. La maquina a utilizarse fue una AMD K6, 500 Mhz y 128 Mb de memoria, utilizando Windows 2000 Profesional. La figura 6.6 nos muestra la configuración del MODEM a utilizarse. Es un HSP56 MicroModem configurado de forma similar a la prueba anterior., esto es, responder automáticamente con la grabadora de mensajes al quinto timbrado, grabar el mensajes con una longitud máxima de 60 segundos, o permitir el acceso remoto a los mensajes con una contraseña (123) al tercer timbrado.

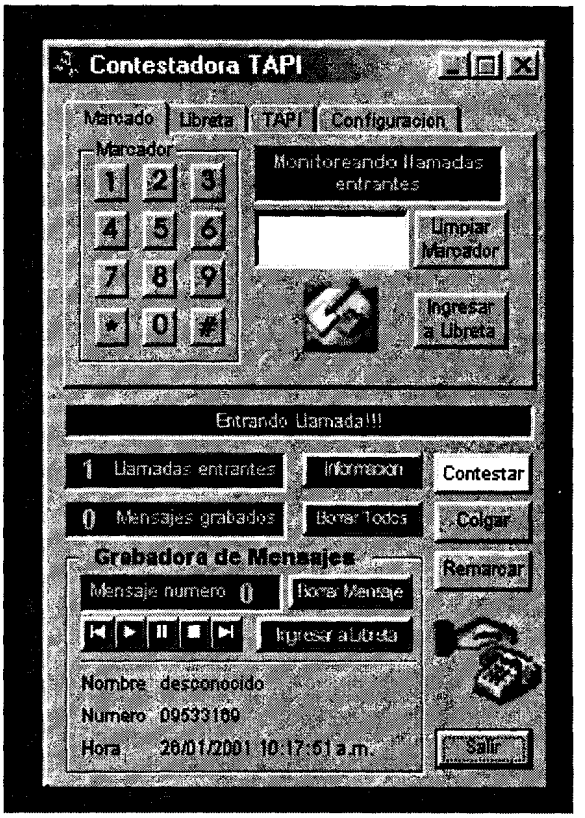

Figura 6.6 Detectando llamada entrante

Podemos constatar como el proceso es similar a la prueba anterior. Sin embargo podemos notar ahora como en el campo nombre sale la palabra desconocido. Esto se debe a que en la trama Caler Id no viene indicado información concerniente al nombre de la persona, por lo que el programa realizara una búsqueda en su base de datos para encontrar un nombre asociado a el numero telefónico. Al no encontrar una asociación, el nombre desconocido es colocado en el campo nombre. Sin embargo, se podría incluir este numero en la base de datos de números telefónicos, y añadir un nombre correspondiente. De esta manera, se encontraría un nombre asociado al teléfono y este seria el utilizado por el programa para futuras llamadas.

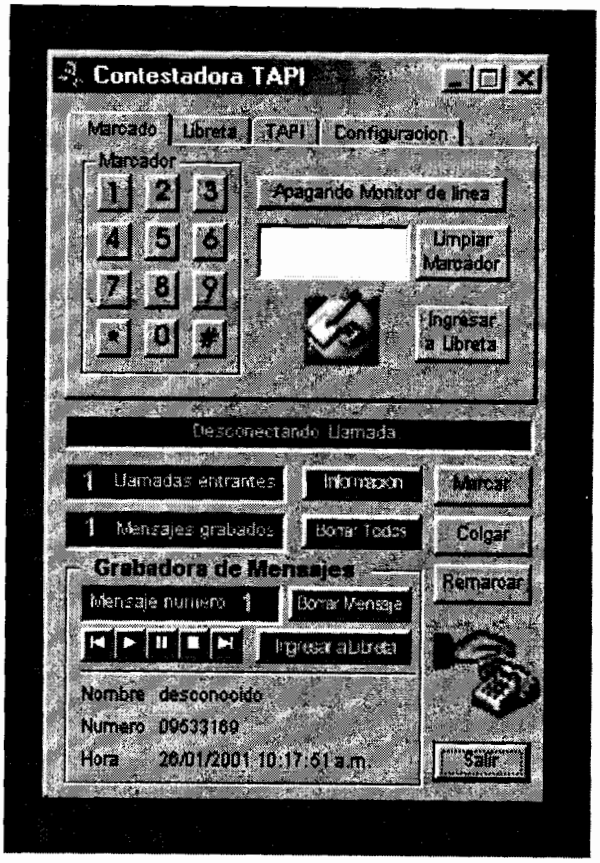

Figura 6.7 Desconectando llamada

Figura 6.8 Comprobación de información Caller Id

6.3 Prueba de llamada desde una localidad en otra provincia

Esta prueba fue realizada desde el mismo hogar en la ciudadela El Paraíso, al día siguiente de la prueba anterior. La máquina y la configuración a utilizarse fue la misma del día y experimento anterior. Las figuras 6.9 y 6.10 nos muestran el comportamiento del programa durante el transcurso de la llamada.

Figura 6.9 Detección de llamada entrante

Podemos apreciar que la llamada entrante corresponde a una persona que vive en la provincia de Pichincha. De igual forma, vemos que se la identifica como persona desconocida, al no tenerse registro de su nombre en la base de datos interna. En la figura 6.11 podemos constatar la misma información Caller Id proporcionada por el programa.

Figura 6.10 Desconexión de llamada

Figura 6.11 Comprobación de información Caller Id

6.4 Resumen

Básicamente debemos separar las pruebas de comprobación en tres posibilidades, las cuales muestran las diferentes posibilidades que un usuario puede afrontar diariamente. Estas pruebas son las de llamada local, llamada de celular y llamada desde otra provincia.

Todas ellas se efectuaron exitosamente en diferentes locales de la ciudad de Guayaquil, en donde se habia contratado el servicio Caller Id en la central telefónica local, en este caso, Pacifictel. En todas estas casas se contaba con un computador con hardware apropiado para las pruebas, en este caso, un MODEM que pueda manejar voz.

Conclusiones y recomendaciones

El trabajo presentado aquí es una investigación teórica para establecer la factibilidad de usar las librerías TAPI para las comunicaciones de baja, mediana y alta velocidad en el ambiente de área local. Esto fue seguido de una demostración de su aplicación, desarrollando un pequeño pero práctico programa telefónico de escritorio que soporte las funciones principales de un teléfono, e incluyendo funciones extendidas, como manejo de libretas telefónicas y, principalmente, soporte para el servicio Caller Id. Basaremos nuestras conclusiones en los resultados obtenidos y en la experiencia al desarrollar y estudiar las funciones proporcionadas por TAPI.

El diseño de aplicaciones utilizando las librerías TAPI proporciona una base de desarrollo de aplicaciones sólida y escalable. No es necesario software de terceros para poder proporcionar funciones profesionales de servicios telefónicos, y en general, de servicios de telefonía y comunicaciones. Su utilización es sencilla y su funcionamiento es amplio. Constituido como uno de los mas importantes estándares para el desarrollo de hardware, se puede desarrollar software basado en TAPI para proporcionar servicios telefónicos sin tener que utilizar características de hardware similares en diferentes localidades.

Al existir actualizaciones de hardware telefónico, no es necesario el realizar cambios en los antiguos módulos del programa. Utilizando TAPI se puede agregar funciones inexistentes antes en el programa, con el fin de

aprovechar las nuevas capacidades del dispositivo telefónico, sin tener que afectar sus antiguas capacidades y sin tener que variar este código.

TAPI no solo proporciona solución es de desarrollo para funciones de usuario final, como se ha descrito en este programa. Además, se puede manejar flujo datos en procesos mas complejos, como en conexiones directas de MODEM y transmisiones directas de datos. Se considera importante dar a conocer esta herramienta para poder explotar su potencial y el potencial de desarrollo que este involucra para las industrias.

Se debe analizar correctamente las capacidades del MODEM que se esta utilizando. Un MODEM puede ser soportado por TAPI, pero puede presentar capacidades limitadas para los servicios soportados por TAPI. Es así que, por ejemplo, un MODEM puede ser soportado por TAPI, pero este puede no estar capacitado para analizar la trama Caller Id. Además, los Wave-MODEM pueden no ser Full-Duplex, además de no tener una unión real con la tarjeta de sonido de la máquina, lo que produciría que se escuchen llamadas, pero no poder hablar por el micrófono.

Se debe estudiar otros sistemas de análisis de tramas de Caller Id. Ante todo, esto dependerá del tipo de trama existente en la localidad, pues esta puede diferir en zonas, he incluso en sistemas telefónicos locales de oficina y los públicos. El sistema actual funciono con las pruebas con la telefonía publica, pero fallo en la recepción en sistemas especializados, como en redes telefónicas privadas y algunas universitarias, donde se habían implementado otros sistemas de caller id para teléfonos dedicados.

Considerando el potencial de las librerías TAPI, se recomienda realizar también un estudio del alcance de otros componentes del sistema WOSA. Una de mis recomendaciones seria, por ejemplo, analizar el alcance de MAPI

y de SAPI. Las librerías de correo y de Voz de WOSA. Particularmente, considera que las utilización de MAPI podría ser muy beneficiosa y practica para el desarrollo de solución es para la pequeña y mediana empresa.

El programa desarrollado es básico, pero funcional. Sin embargo, sus capacidades pueden ser ampliadas para proporcionar mayor alcance para funciones de oficina más extensas. Basándonos en estas premisas y en el estudio realizado, podemos recomendar ampliar la utilidad del programa utilizando, de forma similar a TAPI, algún otro componente de la familia WOSA. Analizaremos algunos ejemplos: Una opción seria la de incorporar, por ejemplo, mensajería, lo cual se realizaría utilizando MAPI. Además, comandos de reconocimiento de voz para realizar tareas simples y directas como reenvío, parqueo o habilitar la contestadora automática podrían ser implementadas, las cuales se desarrollarían utilizando Sapi.

Anexo A

Tabla I Funciones asistidas

Llamado API	Parámetros	Comentarios
TAPIRequestMakeCall	DestAddress, AppName, CalledParty, Comment	Use esta función para demandar una colocación con rumbo exterior de llamada. Sólo los DestAddress son requeridos.
TAPIGetLocationInfo	CountryCode, CityCode	Use esta función para devolver el código de código del país actual y de la ciudad de la estación de trabajo. Estos valores son almacenados en el TELEPHON.INI

Tabla II Funciones de Servicio Básico

Manejo básico de la línea	lineInitialize	Inicializa abstractamente la línea para ser utilizada por la API
	lineShutdown	Termina con la utilización de la línea por parte de la aplicación

	lineNegotiateAPIVersion	Negocia una versión del API para ser utilizado
	lineOpen	Abre la línea para su consiguiente utilización y monitoreo
	lineClose	Cierra una línea abierta
	lineDrop	Desconecta una llamada, e interrumpe cualquier proceso
	lineDeallocateCall	"Descoloca" al manejador actual de la línea
Configuración y estado	lineGetDevCaps	Retorna las capacidades del dispositivo de línea a utilizarse
	lineGetDevConfig	Retorna la configuración del dispositivo
	lineGetlineDevStatus	Retorna el estado de la línea actual.
	linesetDevConfig	Configura el dispositivo de línea
	linesetStatusMessages	Especifica un cambio de estado en una situación predispuesta por la aplicación
	lineGetStatusMessages	Retorna el estado actual de los mensajes y la línea utilizados por la aplicación.
	lineGetID	Obtiene el identificador del dispositivo
	linesetNumRings	Indica el número de timbrados

		para responder llamadas entrantes.
	lineGetNumRings	Retorna el número indicado por linesetNumRings.
	lineGetIcon	Permite a la aplicación obtener un icono representativo de la llamada.
	lineConfigDialog	Permite aparecer una pantalla de configuración de aspectos del proveedor del servicio
Llamadas entrantes y salientes	lineMakeCall	Inicializa una llamada saliente y especifica un manejador para ella
	lineDial	Marca una o varias direcciones.
	lineanswer	Responde una llamada entrante
Direccionamiento	lineGetAddressCaps	Retorna las capacidades de la dirección determinada
	lineGetAddressStatus	Retorna el estado actual de la dirección determinada
	lineGetAddressID	Obtiene el identificador de la dirección cuando este esta en otros formatos
	lineTranslateAddress	Traduce la dirección en forma canónica a una en forma marcable

	linesetCurrentLocation	Especifica la locación para ayudar al proceso de traducción
	linesetTollList	Manipula la lista de llamadas en proceso.
	lineGetTranslateCaps	Retorna las habilidades de traducción de la dirección
Varios	lineGetCallInfo	Retorna información constante sobre la dirección.
	lineGetCallStatus	Retorna información de estado extendida de la dirección.
	linesetAppSpecific	Especifica la estructura de la información
	LineRegisterRequest ÂRecipient	Registra o desregistra a una aplicación como un correcto punto de llegada de tipos de contactos entrantes.
	lineGetRequest	Obtiene el siguiente requerimiento del DLL telefónico
	linesetCallPrivilege	Especifica los privilegios de la llamada.
	lineHandoff	Cambia los privilegios de una aplicación sobre una llamada.
	lineGetNewCalls	Proporciona manejadores de llamadas para llamadas que la aplicación aun no obtiene

		manejadores.
	lineGetConfRelatedCalls	Retorna los manejadores de llamadas que realmente corresponden a una sola llamada tipo conferencia.

Tabla III Estructura de datos de la Api Básica de Telefonía

LINETRANSLATEOUTPUT	Describe el resultado de una traducción de la dirección
LINETRANSLATECAPS	Describe las capacidades de traducción de la dirección
LINETERMCAPS	Describe las capacidades del dispositivo terminal de una línea
LINEREQMAKECALL	Describe una petición del TAPIRequestMakeCall
LINEPROVIDERLIST	Describe una lista de país proveedores de servicios
LINEPROVIDERENTRY	Provee la información para una entrada del proveedor del solo servicio.
LINEMONITORTONE	Describe un tono para ser monitoreado
LINEMEDIACONTROLTONE	Describe una acción de soporte lógico informático para ser ejecutado cuando un tono ha sido detectado
LINEMEDIACONTROLMEDIA	Describe una acción de soporte lógico informático para ser ejecutado al detectar un cambio del modo - soporte lógico informático

LINEMEDIACONTROLDIGIT	Describe una acción de soporte lógico informático para ser ejecutado al detectar un dígito.
LINEMEDIACONTROLCALLSTATE	Describe una acción de soporte lógico informático para ser ejecutado al detectar transiciones en uno o más llamar estados
LINELOCATIONENTRY	Describe que una localización solió proveer un contexto de traducción de la dirección.
LINEGENERATETONE	Contiene información acerca de un tono para ser generado
LÍNEADDRESSCAPS	Describe las capacidades de una dirección especificada.
LÍNEADDRESSSTATUS	Describe el estado actual de una dirección
LINECALLINFO	Contiene información acerca de una llamada
LINECALLLIST	Describe una lista de agarraderas de llamada.
LINECALLPARAMS	Describe parámetros abastecidos al hacer lineMakeCall que usa llamadas
LINECALLSTATUS	Describe el estado actual de una llamada
LINECARDENTRY	Describe una tarjeta de visita
LINECOUNTRYENTRY	Provee la información para una sola entrada del país
LINECOUNTRYLIST	Describe una lista de países
LINEDEVCAPS	Describe las capacidades de un

	dispositivo de línea
LINEDEVSTATUS	Describe el estado actual de un dispositivo de línea
LINEDIALPARAMS	Especifica un grupo de campos relacionados al discado
LINEEXTENSIONID	Describe una CALLER ID de la extensión. Las identificaciones de la extensión se usan para identificar servicio extensiones específicas en proveedor para dispositivos de línea
LINEFORWARD	Describe una entrada de las instrucciones de envío
LINEFORWARDLIST	Describe una lista de instrucciones de envío

Tabla IV Mensajes Básicos

Message	Parameters	Description

LINE_ADDRESSSTATE	dwDevice = hLine;
	dwCallbackInstance = Callback;
	dwParam1 = idAddress;
	dwParam2 = AddressState;
	dwParam3 = (DWORD) 0;
		Enviado cuando el estado de una dirección cambia en una línea que está actualmente abierta por la aplicación. La aplicación puede invocar lineGetAddressStatusto para determinar el estado actual de la

		dirección.

LINE_CALLINFO	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 = CallInfoState;
	dwParam2 = (DWORD) 0;
	dwParam3 = (DWORD) 0;
		Enviado cuando la información acerca de la llamada especificada haya cambiado. La aplicación puede invocar que lineGetCallInfo para determinar la información actual de llamada.

LINE_CALLSTATE	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 = CallState;
	dwParam2 = CallStateDetail;
	dwParam3 = CallPrivilege;
		Enviado cuando el estado de la llamada especificada haya cambiado. Varios de estos mensajes típicamente serán recibidos durante la duración de una llamada.

LINE_CLOSE	dwDevice = hLine;
	dwCallbackInstance = hCallback;
	dwParam1 = (DWORD) 0;
	dwParam2 = (DWORD) 0;
	dwParam3 =
		Enviado cuando el dispositivo especificado de línea se fuerce a cerrarse. El manejador del dispositivo de línea, cualquier agarradera de la llamada o llamadas con retardo en llegar

	(DWORD) 0;	son no válidas una vez este mensaje ha sido enviado.

LINE_CREATE	dwDevice = 0;
	dwCallbackInstance = 0;
	dwParam1 = idDevice;
	dwParam2 = 0;
	dwParam3 = 0;
		Expedido para informar a la aplicación de la creación de un dispositivo nuevo de línea.

LINE_DEVSPECIFIC	dwDevice = hLineOrCall;
	dwCallbackInstance = hCallback;
	dwParam1 = DeviceSpecific1;
	dwParam2 = DeviceSpecific2;
	dwParam3 = DeviceSpecific3;
		Expedido para hacer saber a la aplicación acerca de sucesos específicos en dispositivo ocurriendo en una línea, dirección, o llamada. El significado del mensaje y la interpretación de los parámetros es dispositivo específico

LINE_DEVSPECIFICFEATURE	dwDevice = hLineOrCall;
	dwCallbackInstance = hCallback;
	dwParam1 = DeviceSpecific1;
	dwParam2 =
		Expedido para hacer saber a la aplicación acerca de sucesos específicos en dispositivo ocurriendo en una línea, dirección, o llamada. El

	DeviceSpecific2; dwParam3 = DeviceSpecific3;	significado del mensaje y la interpretación de los parámetros es dispositivo específico.

LINE_GATHERDIGITS	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 = GatherTermination;
	dwParam2 = 0;
	dwParam3 = 0;
		Enviada cuando la petición de recoger dígitos ha terminado o se cancela. El buffer de los dígitos puede ser examinado después de que este mensaje haya sido recibido por la aplicación.

LINE_GENERATE	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 = GenerateTermination;
	dwParam2 = 0;
	dwParam3 = 0;
		Expedido para hacer saber a la aplicación que el dígito actual o la generación de tono termino. Note que tal petición de generación solo puede ser llamada una vez, pero puede pasar en el progreso en una llamada dada en cualquier momento. Este mensaje es también enviado cuando dígito o la generación de tono se cancele..

LINE_LINEDEVSTATE	dwDevice = hLine;
	dwCallbackInstance = hCallback;
	dwParam1 = DeviceState;
	dwParam2 = DeviceStateDetail1;
	dwParam3 = DeviceStateDetail2
		Enviado cuando el estado de un dispositivo de línea haya cambiado. La aplicación puede invocar que lineGetLineDevStatus determine el estado nuevo de la línea.

LINE_MONITORDIGITS	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 = Digit;
	dwParam2 = DigitMode;
	dwParam3 = 0;
		Enviado cuando un dígito sea detectado. El envío de este mensaje está controlado con la función de lineMonitorDigits.

LINE_MONITORMEDIA	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 = MediaMode;
	dwParam2 = 0;
	dwParam3 = 0;
		Enviado cuando un cambio en el tipo de medio de la llamada sea detectado. El envío de este mensaje está controlado con la función del lineMonitorMedia.

LINE_MONITORTONE	dwDevice = hCall;
	dwCallbackInstance = hCallback;
	dwParam1 =
		Enviado cuando un tono sea detectado. El envío de este mensaje está controlado con la

	dwAppSpecific;
	dwParam2 = 0;
	dwParam3 = 0;
		función de lineMonitorTones.

LINE_REPLY	dwDevice = 0;
	dwCallbackInstance = hCallback;
	dwParam1 = idRequest;
	dwParam2 = Status;
	dwParam3 = 0;
		Expedido para reportar los resultados de llamados que terminan asincrónicamente.

LINE_REQUEST	dwDevice = 0;
	dwCallbackInstance = hRegistration;
	dwParam1 = RequestMode;
	dwParam2 = RequestModeDetail1;
	dwParam3 = RequestModeDetail2;
		Expedido para reportar la llegada de una petición nueva de otra aplicación

Tabla V Funciones Suplementarias

Función	Llamado API	Descripción
Manejos de dígitos y tono	lineMonitorDigits	Habilita o deshabilita la notificación de detección de dígito en una llamada especificada.

	LineGatherDigits	Realiza un Buffer de dígitos en una llamada
	LineMonitorTones	Especifica cuáles tonos detectar en una llamada especificada.
	LineGenerateDigits	Genera dígitos en una llamada.
	LineGenerateTone	Genera un grupo dado de tonos en una llamada.
Manejo avanzado de llamadas	Líneaccept	Acepta una llamada ofrecida y alerta a la persona que llama (ring-back) y al llamado (ring).
	LineRedirect	Reencauza una llamada ofrecida para otra dirección.
	LineSecureCall	Asegura una llamada existente de interferencias por otros sucesos, como pips de llamadas en espera en conexiones de datos.
	LineCompleteCall	Coloca una petición de terminación de llamada.
	LineUncompleteCall	Revoca una petición de terminación de llamada.
Holding	LineHold	Coloca la llamada especificada en agarre

		duro.(Hard Hold)
	LineUnhold	Recupera una llamada sujetada.
Tranferencia	linesetupTransfer	Prepara una llamada especificada para la transferencia a otra dirección.
	LineCompleteTransfer	Transfiere una llamada que fue establecida para la transferencia a otra dirección, o la introduce una conferencia.
	LineBlindTransfer	Transfiere una llamada para otra dirección.
	LineSwapHold	Intercambia la llamada activa con la llamada actualmente en el agarre de consulta.
Conferencia	linesetupConference	Prepara una llamada dada para adicionarla a otra dirección.
	LinePrepareAddToConference	Se dispone a añadir un usuario para una conferencia telefónica existente ubicando una llamada de consulta que más tarde puede agregarse para la

		conferencia telefónica que es colocada en el agarre de conferencia.
	LíneaddToConference	Añade una llamada de consulta para una conferencia telefónica existente.
	LineRemoveFromConference	Remueve un usuario de una conferencia telefónica.
Parking	LinePark	Estaciona un llamado dado en otra dirección.
	LineUnpark	Recupera una llamada estacionada..
Forward	lineForward	Define o revoca requerimientos de reenvío (forward)
Responder	linePickup	Adquiere una llamada que esta en alerta en otro número. Adquiere una llamada alertando en otra dirección de destino y devuelve una agarradera de llamada para la llamada escogida
Miscelaneos	lineSendUserUserInfo	Envía la información de usuario a usuario al usuario remoto en la

		llamada especificada (en ISDN sólo).
	LinesetTerminal	Especifica el dispositivo terminal para el cual los sucesos de línea, la dirección, o los sucesos de soporte lógico informático de llamada son encaminados.
	LinesetCallParams	Demanda un cambio en los parámetros de llamada de una llamada existente.
	LineMonitorMedia	Habilita o deshabilita notificación de tipo de medio en una llamada especificada.
	LinesetMediaControl	Determina el medio de comunicación de la llamada para el debido control.
	LinesetMediaMode	Determina el tipo de medio (s) de la llamada especificada en su estructura LINECALLINFO.

Tabla VI Funciones Extendidas

grupo de función	Llamado	Descripción
servicio extendido de línea	lineNegotiateExtVersion	Permite una aplicación usar con lo negocia una versión de la extensión para dispositivo especificado de línea..
	LineDevSpecific	La función específica en dispositivo de escapada.
	lineDevSpecificFeature	La función específica en dispositivo de escapada a permitir enviar interruptor presenta para el interruptor
servicio Telefónico extendido	phoneNegotiateExtVersion	Permite una aplicación negociar una versión de la extensión para usar con el dispositivo telefónico especificado.
	phoneDevSpecific	La función específica en dispositivo de escapada a dejarle a vendedor-proveedor las extensiones dependientes.

Tabla VII Funciones suplementarias de dispositivos telefónicos

Grupo	Llamado API	Descripción
El manejo	phoneInitialize	Inicializa el dispositivo del

telefónico básico		teléfono de API de Telefonía para el uso por la aplicación invocadora.
	phoneShutdown	Cierra el uso de la aplicación del API de Telefonía.
	phoneNegotiateAPIVersion	Permite una aplicación negociar una versión de API para usar.
	phoneOpen	Abre el dispositivo telefónico especificado, dando la aplicación ya sea dueño o privilegios de monitor.
	PhoneClose	Cierra un dispositivo telefónico abierto especificado.
	PhonesetRing	Marca en un dispositivo telefónico abierto según un modo dado del ring.
	PhoneGetRing	Devuelve el modo actual del ring de un dispositivo telefónico abierto.
Configuración y status	phoneGetDevCaps	Devuelve las capacidades de un dispositivo telefónico dado
	PhoneGetID	Retorna un ID del dispositivo asociada con el

		artificio telefónico especificado.
	PhoneGetIcon	Permite una aplicación recuperar un icono para mostrarlo al usuario.
	PhoneConfigDialog	Hace que el proveedor del dispositivo telefónico especificado presente una caja de diálogo que permite al usuario configurar parámetros en relación con el dispositivo telefónico
	PhonesetStatusMessages	Especifica el estado en el cuál la aplicación quiere ser notificada del suceso.
	PhoneGetStatusMessages	Devuelve los cambios de estado para los cuales la aplicación quiere ser notificada
	PhoneGetStatus	Devuelve el estado completo de un dispositivo telefónico abierto..
	PhonesetHookSwitch	Determina el modo de interruptor de uno o más de los dispositivos del interruptor de gancho de un dispositivo abierto.
	PhoneGetHookSwitch	Busca el modo de

		interruptor de un dispositivo telefónico abierto.
	PhonesetVolume	Determina el volumen de un dispositivo telefónico abierto.
	PhoneGetVolume	Devuelve el volumen colocando en un dispositivo telefónico abierto.
	PhonesetGain	Determina la ganancia del micrófono de un dispositivo telefónico abierto.
	PhoneGetGain	Devuelve la ganancia colocando del mic de un dispositivo Telefónico abierto.
Displays, datos, botones y lámparas	phonesetDisplay	Escribe información para el despliegue de un dispositivo telefónico abierto.
	PhoneGetDisplay	Devuelve los contenidos actuales de despliegue de un teléfono.
	PhonesetButtonInfo	Determina la información asociada con un botón en un dispositivo telefónico.
	PhoneGetButtonInfo	La información de ingreso asociada con un botón en un dispositivo telefónico.

	PhonesetLamp	Ilumina una lámpara en un dispositivo telefónico abierto especificado en un modo que iluminado de lámpara dado.
	PhoneGetLamp	Devuelve el modo lampante actual de la lámpara especificada.
	PhonesetData	Hace un download de un buffer de datos a un área dada de datos en el dispositivo telefónico.
	PhoneGetData	Envía los contenidos de un área dada de datos en el dispositivo telefónico a un Buffer.

Tabla VIII Estructuras básicas del dispositivo telefónico

Estructura	Descripción
PHONEBUTTONINFO	Contiene información acerca de un botón en un dispositivo telefónico..
PHONECAPS	Describe las capacidades de un dispositivo telefónico.
PHONEEXTENSIONID	Describe el ID de la extensión. Los IDs de la extensión se usan para identificar servicios de extensiones específicas para las clases telefónicas del

	dispositivo. Usado en su mayor parte para la Telefonía Extendida.
PHONESTATUS	Describe el estado actual de un dispositivo telefónico.
VARSTRING	Usado para retorno de información de dimensiones variables. Es usado por el dispositivo de línea y el dispositivo telefónico.

Tabla IX Mensajes del dispositivo telefónico

Mensaje	Parámetros	descripción

PHONE_BUTTON	hPhone = hPhoneDevice;
	dwCallbackInstance = hCallback;
	dwParam1 = idButtonOrLamp;
	dwParam2 = ButtonMode;
	dwParam3 = ButtonState;
		Expedido para hacer saber a la aplicación que el monitoreo de los botones esta habilitado, si se ha detectado el uso de un botón en el teléfono local..

PHONE_CLOSE	hPhone = hPhoneDevice;
	dwCallbackInstance = hCallback;
	dwParam1 = 0;
	dwParam2 = 0;
	dwParam3 = 0;
		Enviado cuando un dispositivo telefónico abierto sea forzadamente cerrado como parte de un reclamo de recurso. El nombre de usuario del dispositivo es ya no válido una vez este

		mensaje ha sido enviado.

PHONE_CREATE	hPhone = hPhoneDev;
	dwCallbackInstance = 0;
	dwParam1 = idDevice;
	dwParam2 = 0;
	dwParam3 = 0;
		Este mensaje es enviado para informar a las aplicaciones de la creación de un dispositivo telefónico nuevo.

PHONE_DEVSPECIFIC	hPhone = hPhoneDevice;
	dwCallbackInstance = hCallback;
	dwParam1 = DevSpecific1;
	dwParam2 = DevSpecific2;
	dwParam3 = DevSpecific3;
		Este mensaje es enviado para hace saber a la aplicación acerca de los sucesos específicos en el dispositivo. El significado del mensaje y la interpretación de los parámetros está definido por el vendedor-proveedor del hardware.

PHONE_REPLY	hPhone = 0;
	dwCallbackInstance = hCallback;
	dwParam1 = idRequest;
	dwParam2 = Status;
	dwParam3 = 0;
		Este mensaje es enviado para reportar los resultados de una llamada de función que se completó asincrónicamente.

PHONE_STATE	hPhone = hPhoneDevice; dwCallbackInstance =	El proveedor de servicios envía este

	hCallback;
	dwParam1 = PhoneState;
	dwParam2 =
	PhoneStateDetails;
	dwParam3 = 0;
		mensaje a la función de recuperación (callback) de una aplicación cuandoquiera que el estado de un dispositivo telefónico cambia.

Anexo B

FrmMain.frm

```
Option Explicit
Dim hInst As Long
Dim lineApp As Long
Dim lines As Long
Dim lAdd As Boolean ' agregar a libreta-flag
Dim lMonitor As Boolean ' monitoreo-flag
Dim lDevErr As Boolean ' line device seleccionada-flag
Dim iWidth As Integer ' tamaño del frame
Dim iHeight As Integer ' altura del frame
Public CleanUp As Long          'Resetea despues de grabar*** Timer1 ***
Private g_cMsgs As New Collection   'del sdk..coleccion de wavs de mensajes..
Private g_lMsgPlaying As Long      'chequea si esta sonando algun mensaje
---------------------------------
Private Sub back_Click()

If (nummen - 1) < 1 Then Exit Sub
   nummen = nummen - 1
   lblMsgCount2.Caption = nummen
End Sub
---------------------------------
Private Sub borrar_Click()
      If nummen = 0 Then Exit Sub
      Dim oColl As New Collection
      "Cut Here
      Dim oVar As Variant
      Dim oFiles As New clsFile
      Dim rc As Long
      Set oColl = oFiles.GetMessages
      Kill App.Path & "\Messages\" & oColl.Item(nummen)
      Set oColl = oFiles.GetMessages
      Set oFiles = Nothing
      g_lMsgPlaying = 0
      nummen = 0
      lblMsgCount2.Caption = nummen
      cmdPlay2.Caption = "&Play"
      cmdPlay(0).Visible = True
      cmdRepeat.Visible = False
      StopPlaying
      GetFiles
End Sub
---------------------------------
Private Sub borrart_Click()
      If lblMsgCount.Caption = 0 Then Exit Sub
      Dim oColl As New Collection
      Dim oVar As Variant
      Dim oFiles As New clsFile
      Dim rc As Long
      Set oColl = oFiles.GetMessages
         For rc = 1 To lblMsgCount.Caption
            Kill App.Path & "\Messages\" & oColl.Item(rc)
         Next
      Set oColl = oFiles.GetMessages
      lblMsgCount.Caption = oColl.Count
      Set oFiles = Nothing
      g_lMsgPlaying = 0
      nummen = 0
```

```
      lblMsgCount2.Caption = nummen
      Set g_cMsgs = Nothing
      cmdPlay2.Caption = "&Play"
      cmdPlay(0).Visible = True
      cmdRepeat.Visible = False
      lblName.Caption = ""
      lblNumber.Caption = ""
      LblTime.Caption = ""
      StopPlaying
End Sub
---------------------------------
Private Sub btagrega_Click()
If lblName = "" Then Exit Sub
frmData.txtName.Text = lblName
frmData.txtPhoneNumber.Text = lblNumber
   If frmData.txtPhoneNumber.Text = "" Then
   gName = MsgBox("Ingrese un Numero por favor...", vbOK)
   Exit Sub
   End If
frmData.Show vbModal
End Sub
---------------------------------
Private Sub Check1_Click()
Select Case Check1.Value
   Case 0
      Text1.Visible = False
   Case 1
      Text1.Visible = True
End Select
End Sub
---------------------------------
Public Sub cmdButtons_Click(Index As Integer)
Dim x As Long
Select Case Index
      Case 1
         ansnow = True
         Select Case SSTab1.Tab
            Case 0 ' marcador
               gDialString = txtDialString
                  If gDialString = "" Then
                  gName = MsgBox("Ingrese un Numero por favor...", vbOK)
                  Exit Sub
                  End If
               gName = "<Desconocido>"
               frmCall.Show vbModal
               If gPlaceCall = True Then
                  TAPIPlaceCall (gDialString)
               End If
            Case 1
               gDialString = llamara.Text
                  If gDialString = "" Then
                  gName = MsgBox("Seleccione un Numero por favor...", vbOK)
                  Exit Sub
                  End If
               x = DBGrid1.Row
               gName = DBGrid1.TextMatrix(x, 0)
               frmCall.Show vbModal
               If gPlaceCall = True Then
                  TAPIPlaceCall gDialString
               End If
            Case 2
'nada
            Case 3
'nada
         End Select
      Case 2
```

```
        frmMain.lblName.Caption = ""
        frmMain.lblNumber.Caption = ""
        Label5.Visible = True
        Label5.Caption = "Desconectando Llamada.."
        ansnow = False
        TAPIHangUp
        'TAPIClearHandles
        If lMonitor = True Then
           cmdOnOff_Click  ' apago monitor
           cmdOnOff_Click  ' lo reinicio
        End If
        Label5.Visible = False
        Label5.Caption = "Entrando Llamada!!!"
        frmMain.cmdButtons(1).Visible = True
        frmMain.Command1.Visible = False
     Case 3
        Unload Me
  End Select
End Sub
---------------------------------
Private Sub cmdData_Click(Index As Integer)
 Dim x As Integer
  Select Case Index
     Case 0
      frmData.Show vbModal
     lAdd = True
  Case 1
  Dim iAns As Integer
  iAns = MsgBox("Listo para limpiar Libreta...!", vbInformation + vbYesNo, "Limpiar Libreta")
  If iAns = vbYes Then
     MousePointer = vbHourglass
     Data1.Recordset.MoveFirst
     Do Until Data1.Recordset.EOF
        Data1.Recordset.Delete
        Data1.Recordset.MoveNext
     Loop
     MousePointer = vbNormal
     MsgBox "Libreta Limpiada!!!", vbInformation
  End If
     Data1.Refresh
     lAdd = False
     Case 2
          Data1.Refresh
          lAdd = False
      Case 3 ' borra solo un dato
Dim xx As Long
Dim y As Long
xx = DBGrid1.Row
y = DBGrid1.Col
gName = DBGrid1.TextMatrix(xx, y)
Data1.Recordset.MoveFirst ' comienza por arriba
Do Until Data1.Recordset.EOF
   If (Data1.Recordset.Fields("Telefono") = llamara.Text) And (Data1.Recordset.Fields("Nombre") = gName) Then
     Data1.Recordset.Delete
  Else
 'nada por ahora
  End If
  Data1.Recordset.MoveNext ' voy al siguiente..
Loop
  Data1.Refresh
End Select
  For x = 0 To 2
     cmdData(x).Enabled = True
  Next x
End Sub
---------------------------------
```

```
Private Sub cmdDialPad_Click(Index As Integer)
   Select Case Index
      Case 0 ' remarcar el ultimo
                ansnow = True
             gDialString = ultnumero
                If gDialString = "" Then
                gName = MsgBox("Ingrese un Numero por favor...", vbOK)
                Exit Sub
                End If
             gName = ultnombre
             frmCall.Show vbModal
             If gPlaceCall = True Then
                TAPIPlaceCall (gDialString)
             End If
      Case 1 ' limpia dialstring
         txtDialString = ""
      Case 2 ' agrega un nuevo valor a la libreta de telefonos
         frmData.txtPhoneNumber.Text = txtDialString
         cmdData_Click 0 ' agrega
         'SSTab1.Tab = 1 ' muestra pagina
   End Select
End Sub
---------------------------------
Private Sub cmdGreeting_Click()
   dlgGreeting.Show vbModal
End Sub
---------------------------------
Public Sub myInit(fSetup As Boolean)
Dim nError As Long
Dim lpExtensionID As lineextensionid
Dim lUnused As Long
Dim lLineID As Long
Dim iNumDev As Long
Dim lRtn As Long
Dim iLoop As Integer
   lLineID = GetSetting("VB-TAPI", "Settings", "DeviceID", "0")
   AddText Me.Text1, m_TapiInit & ":inicio de TAPI - Inicio myInit"
   frmMain.AddText frmMain.Text1, m_TapiInit & " = m_TapiInit Inicio myInit"
   If m_TapiInit = False Then
      DoEvents
      nError = lineInitialize(hTAPI, App.hInstance, _
               AddressOf LineCallBack, 0, lNumLines)
      If nError <> 0 Then
         ProcessTAPIError nError
         AddText Me.Text1, TapiErrMsg(nError) & "Can not initialize TAPI"
      Else
         AddText Me.Text1, "lineInitialize -> Success"
         DebugString 4, "lineInitialize -> Success"
      End If
Setuptruco:
      nError = lineNegotiateAPIVersion(hTAPI, lLineID, TAPIVERSION, _
         TAPIVERSION, lNegVer, lpExtensionID)
AddText Me.Text1, "Chequeando Dev #" & CStr(lLineID) & "...."
AddText Me.Text1, "Line Negociation: Dev #" & CStr(lLineID) & " soporta ver " & Hex(lNegVer)
      If nError <> 0 Then
         AddText Me.Text1, TapiErrMsg(nError) & " Error en tarjeta-device" & lLineID & " !!"
         If fSetup = True Then
            lLineID = lLineID + 1
            If lLineID <= lNumLines Then GoTo Setuptruco
         End If
         ProcessTAPIError nError
         AddText Me.Text1, TapiErrMsg(nError) & "Init TAPI Error: No puede negociar minima version TAPI 1.4"
      Else
         If fSetup = True Then Exit Sub
         nError = lineOpen(hTAPI, lLineID, hLine, lNegVer, lUnused, lUnused, _
```

```
            LINECALLPRIVILEGE_OWNER + LINECALLPRIVILEGE_MONITOR,
LINEMEDIAMODE_INTERACTIVEVOICE + LINEMEDIAMODE_AUTOMATEDVOICE, 0)
        linehandler = hLine
        If nError <> 0 Then
        ProcessTAPIError nError
        AddText Me.Text1, TapiErrMsg(nError) & "lineOpen -> UnSuccess"
        Else
        AddText Me.Text1, "lineOpen -> Success"
        End If
      End If
      lpLineDevCaps.dwTotalSize = Len(lpLineDevCaps)
      nError = lineGetDevCaps(hTAPI, lLineID, lNegVer, lUnused, lpLineDevCaps)
      If nError <> 0 Then
        ProcessTAPIError nError
        AddText Me.Text1, TapiErrMsg(nError) & " - lineGetDevCaps"
        Else
        AddText Me.Text1, "Success - lineGetDevCaps"
      End If
      nError = lineSetStatusMessages(hLine, lpLineDevCaps.dwLineStates, 0)
      If nError <> 0 Then
        ProcessTAPIError nError
        AddText Me.Text1, TapiErrMsg(nError) & "Init TAPI : No puede configurar mensajes de estado.."
      End If
      m_TapiInit = True
      DebugString 3, "m_TapiInit=" & CStr(m_TapiInit)
      cmdOnOff.Caption = "Apagando Monitor de linea"
      cmdOnOff.Visible = False
      lblMonitor.Visible = True
      lMonitor = True
   Else   'apago a Tapi..
      If hTAPI <> 0 Then
        nError = lineShutdown(hTAPI)
        If nError <> 0 Then ProcessTAPIError nError
        AddText Me.Text1, "MyInit : Servicio Tapi apagado!!"
        hTAPI = 0
        m_TapiInit = False
        cmdOnOff.Caption = "Iniciar Monitoreo De Linea"
        cmdOnOff.Visible = True
        lblMonitor.Visible = False
        lMonitor = False
      End If
   End If
AddText Me.Text1, m_TapiInit & ":Flag de inicio de TAPI - Final de myInit"
frmMain.AddText frmMain.Text1, m_TapiInit & " = m_TapiInit Final myInit"
End Sub
---------------------------------
Private Sub cmdKey_Click(Index As Integer)
   Dim cDigit As String
   cDigit = Mid("123456789*0#", Index + 1, 1)
   txtDialString = txtDialString & cDigit ' los pone en la libreta de telefonos.. hay que cambiar de tab para poder
verlo..
End Sub
---------------------------------
Private Sub cmdOnOff_Click()
On Error GoTo EH
   Screen.MousePointer = vbHourglass
   myInit False
   Screen.MousePointer = vbNormal
   Exit Sub
EH:
Screen.MousePointer = vbNormal
MsgBox err.Number & " : " & err.Description
End Sub
---------------------------------
Private Sub GetFiles()
On Error GoTo EH
```

```
Dim oFs As New clsFile
   Set g_cMsgs = oFs.GetMessages
   lblMsgCount.Caption = g_cMsgs.Count
   If g_cMsgs.Count > 0 Then SetCallerInfo (g_cMsgs.Item(g_cMsgs.Count))
   Exit Sub
EH:
MsgBox err.Number & ": " & err.Description
End Sub
---------------------------------
Private Sub cmdPlay_Click(Index As Integer)
If m_TapiInit = True Then
   myInit False
   End If
lMediaID = -1   'The wave mapper, -1 or WAVE_MAPPER as a device id selects
            'the default wave device on the system that can handle the
            'wave format, usually, and hopefully the sound card.
DoPlay
End Sub
---------------------------------
Public Sub DoPlay()
If cmdPlay2.Caption = "&Pause" Then  'was playing, going to paused
   PausePlay
   cmdPlay2.Caption = "&Paused"
   cmdRepeat.Visible = False
   cmdPlay(0).Visible = True
   Exit Sub
End If
If cmdPlay2.Caption = "&Paused" Then 'was paused, going to playing (pause caption)
   ResumePlay
   cmdPlay2.Caption = "&Pause"
   cmdRepeat.Visible = True
   cmdPlay(0).Visible = False
   Exit Sub
End If
Dim oFiles As New clsFile
Set g_cMsgs = oFiles.GetMessages
Set oFiles = Nothing
   If g_cMsgs.Count > 0 Then
      If nummen = 0 Then nummen = 1
      g_lMsgPlaying = nummen
      cmdPlay2.Caption = "&Pause"
      cmdRepeat.Visible = True
      cmdPlay(0).Visible = False
      PlayMessages
   End If
End Sub
---------------------------------
Private Sub cmdStartup_Click(Index As Integer)
   Select Case Index
      Case 0 ' aplica y graba valores..
         gMinimize = chkMinimize
         gmonitor = chkMonitor
         gOutLog = chkOutLog
         gOrigNumber = txtOrigNumber.Text
         gStartPage = IIf(optStartup(0) = True, 0, 1)
         SaveSetting App.EXEName, "Startup", "MinimizeOnStart", gMinimize
         SaveSetting App.EXEName, "Startup", "Monitor", gmonitor
         SaveSetting App.EXEName, "Startup", "OutLog", gOutLog
         SaveSetting App.EXEName, "Startup", "OrigNumber", gOrigNumber
         SaveSetting App.EXEName, "Startup", "StartPage", gStartPage
      Case 1 ' Cancelar
         ' nada
   End Select
End Sub
---------------------------------
Private Sub cmdStop_Click()
```

```
StopPlaying
End Sub
---------------------------
Public Sub StopPlaying()
Dim oFSO As New clsFile
Dim myColl As New Collection
On Error GoTo EH
   StopPlay
   g_lMsgPlaying = -1  'Try not to erase the messages if stop was clicked.
   If nummen > 0 Then nummen = nummen - 1
   Set g_cMsgs = Nothing
   Set myColl = oFSO.GetMessages
   lblMsgCount.Caption = CStr(myColl.Count)
   lblMsgCount2.Caption = nummen
   cmdPlay2.Caption = "&Play"
   cmdPlay(0).Visible = True
   cmdRepeat.Visible = False
   Set myColl = Nothing
   Set oFSO = Nothing
   Exit Sub
EH:
DebugString 0, err.Number & " : " & err.Description
End Sub
---------------------------------
Private Sub PlayMessages()
On Error GoTo EH
   g_lMsgPlaying = nummen
   If g_lMsgPlaying > lblMsgCount.Caption Then
      Dim oColl As New Collection
      "Cut Here
      Dim oVar As Variant
      Dim oFiles As New clsFile
      Dim rc As Long
      Set oColl = oFiles.GetMessages
      lblMsgCount.Caption = oColl.Count
      Set oFiles = Nothing
      g_lMsgPlaying = 0
      nummen = 0
      lblMsgCount2.Caption = nummen
      Set g_cMsgs = Nothing
      cmdPlay2.Caption = "&Play"
      cmdPlay(0).Visible = True
      cmdRepeat.Visible = False
      StopPlaying
      Exit Sub
   End If
   If g_lMsgPlaying = 0 Then Exit Sub  '=0 when stop has been clicked.
   lblMsgCount2.Caption = CStr(g_lMsgPlaying)
   SetCallerInfo CStr(g_cMsgs.Item(g_lMsgPlaying))
   If m_BPlayRec <> False Then
      If (fPlaying = False) Then
         LoadFile App.Path & "\Messages\" & g_cMsgs.Item(g_lMsgPlaying)
         Play lMediaID
         m_BPlayRec = True
      End If
   End If
   nummen = nummen + 1
   Exit Sub
EH:
DebugString 0, err.Number & " : " & err.Description
End Sub
---------------------------
Private Sub SetCallerInfo(sInfo As String)
Dim lPos As Long
On Error Resume Next
   lPos = InStr(1, sInfo, "~")
```

```
    lblName.Caption = Mid(sInfo, 1, lPos - 1)
    sInfo = Mid(sInfo, lPos + 1, Len(sInfo))
    lPos = InStr(1, sInfo, "~")
    lblNumber.Caption = Mid(sInfo, 1, lPos - 1)
    sInfo = Mid(sInfo, lPos + 1, Len(sInfo))
    LblTime.Caption = Mid(sInfo, 1, Len(sInfo) - 4)
End Sub
---------------------------------
Private Sub cmdRepeat_Click()
    StopPlay
    nummen = nummen - 1
End Sub
Private Sub cmdSetup_Click()
myInit False
    dlgSetup.Show vbModal
End Sub
---------------------------------
Private Sub Command1_Click()
ansnow = True
Answer
End Sub
---------------------------------
Private Sub Command2_Click()
    frmbases.Show vbModal
End Sub
---------------------------------
Private Sub Command6_Click()
    Dim retcode As Long
    Dim lngDev As String
    lngDev = 0
    lngDev = InputBox("Ingrese Line Device ID# (Valor Predeterminado = 0): ", "lineConfigDialog", lngDev)
    If lngDev = "" Then GoTo final
    retcode = lineConfigDialog(lngDev, Me.hWnd, "comm")
    If retcode <> 0 Then
        AddText Me.Text1, TapiErrMsg(retcode) & " - lineConfigDialog"
    End If
final:
End Sub
---------------------------------
Private Sub DBGrid1_Click()
Dim x As Long
x = DBGrid1.Row
llamara.Text = DBGrid1.TextMatrix(x, 1)
End Sub
---------------------------------
Private Sub DBGrid1_DblClick()
Dim x As Long
Dim y As Long
x = DBGrid1.Row
y = DBGrid1.Col
llamara.Text = DBGrid1.TextMatrix(x, y)
Select Case y
    Case 0
gName2 = DBGrid1.TextMatrix(x, 1)
gName = InputBox("Cambiar nombre a...")
If gName = "" Then GoTo mefui
Data1.Recordset.MoveFirst ' comienza por arriba
Do Until Data1.Recordset.EOF
    If (Data1.Recordset.Fields("Nombre") = llamara.Text) And (Data1.Recordset.Fields("Telefono") = gName2) Then
        Data1.Recordset.Edit
        Data1.Recordset.Fields("Nombre") = gName
        Data1.Recordset.Update
    Else
 'nada por ahora
    End If
    Data1.Recordset.MoveNext ' voy al siguiente..
```

```
Loop
   Data1.Refresh
   Case 1
gName2 = DBGrid1.TextMatrix(x, 0)
gName = InputBox("Cambiar Telefono ...")
If gName = "" Then GoTo mefui
Data1.Recordset.MoveFirst ' comienza por arriba
Do Until Data1.Recordset.EOF
    If (Data1.Recordset.Fields("Telefono") = llamara.Text) And (Data1.Recordset.Fields("Nombre") = gName2) Then
      Data1.Recordset.Edit
      Data1.Recordset.Fields("Telefono") = gName
      Data1.Recordset.Update
   Else
 'nada por ahora
   End If
   Data1.Recordset.MoveNext ' voy al siguiente..
Loop
   Data1.Refresh
End Select
mefui:
llamara.Text = DBGrid1.TextMatrix(x, 1)
End Sub
---------------------------------
Private Sub Form_Unload(Cancel As Integer)
On Error Resume Next
   cmdStartup_Click 0
   Unload dlgGreeting
   Unload dlgSetup
   Unload frmbases
   Unload frmData
   Unload frmCall
   TAPIHangUp  ' cuelga cualquier linea dejada abierta
   TAPIShutdown ' cierra Tapi
End Sub
---------------------------------
Private Sub forward_Click()
   If (nummen + 1) > lblMsgCount.Caption Then Exit Sub
   If cmdPlay2.Caption <> "&Play" Then
   StopPlay
   Else
   nummen = nummen + 1
   g_lMsgPlaying = nummen
   lblMsgCount2.Caption = CStr((g_lMsgPlaying))
   End If
End Sub
---------------------------------
Private Sub pause_Click()
If cmdPlay2.Caption = "&Pause" Then  'was playing, going to paused
   PausePlay
   cmdPlay2.Caption = "&Paused"
   cmdRepeat.Enabled = False
   Exit Sub
End If
If cmdPlay2.Caption = "&Paused" Then 'was paused, going to playing (pause caption)
   ResumePlay
   cmdPlay2.Caption = "&Pause"
   cmdRepeat.Enabled = True
   Exit Sub
End If
End Sub
---------------------------------
Private Sub Timer1_Timer()
On Error GoTo EH
   If sSecret = dlgSetup.txtSecret Then
      sSecret = ""
      CloseWaveOut
```

```
    fInTollSaver = True
    DoEvents
    DoPlay
  End If
  lTollSaver = lTollSaver + 1
  If lTollSaver > 10 Then 'prueba 1 sigue aqui *********
    lTollSaver = 0     'Reset. lTollSaver = 0 en Globals.bas
    sSecret = ""       'siempre que un digito es detectado.
  End If
  If m_BPlayRec = False Then  'grabando...prueba******inicio***
    If CleanUp = 1 Then     'grabado fin...prueba****fin callback
        Dim nErr As Long 'proc. waveInProc en Wave.bas SDK
        DebugString 5, "CleanUp in Timer"
        CloseWaveIn
        Timer1.Enabled = False  'prueba 1 sigue aqui..ojo**
        SaveToFileAsStream m_FileName
        m_BPlayRec = True   'ya no estamos grabando mas aqui...
        If dlgGreeting.Visible = False Then
          lblMsgCount.Caption = lblMsgCount.Caption + 1
          DropCall
          hCall = 0
          CleanUp = 0 'prueba 1..dejar esta linea aqui..
        End If
    End If
  Else
    If fPlaying = False Then    'como waveOutProc en Wave.bas SDK
      DebugString 5, "fPlaying shutting down wave out device"
      CloseWaveOut
      Timer1.Enabled = False
      If hmem <> 0 Then
        GlobalFree (hmem)
        hmem = 0
      End If
      If m_TapiInit = True Then   'Tapi inicio***
        CleanUp = 0
        If fInTollSaver = False Then
          m_BPlayRec = False
          SendDigit
        Else
          PlayMessages    'prueba de mensajeria remota 1**
        End If
      Else
        If dlgGreeting.Visible = False Then
          PlayMessages    'prueba de mensajeria remota 1**
        End If
      End If
    End If
  End If 'm_BPlayRec = False prueba**
  Exit Sub
EH:
DebugString 0, err.Number & " : " & err.Description
End Sub
---------------------------------
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
On Error Resume Next
  If Timer1.Enabled = True Then   'no cierra si esta sonando***
    Cancel = 1
  Else
    cmdStop_Click
    If m_TapiInit = True Then myInit False
  End If
End Sub
---------------------------------
Private Sub Form_Load()
  CleanUp = 0
  m_BPlayRec = True
```

```
    g_lMsgPlaying = 0
    Me.Top = (Screen.Height / 2) - (Me.Height / 2)
    Me.Left = (Screen.Width / 2) - (Me.Width / 2)
    ansnow = False
    On Error Resume Next    'prueba de errores...crear directorio de mensajes...
    MkDir "Messages"
    GetFiles
       On Error GoTo LocalErr
    Dim lRtn As Long
    Dim cMsg As String
    iWidth = Me.Width
    iHeight = Me.Height
    InitDB
    ReadStartup ' carga configuracion del registro
    If gLineDev = -1 Then
    SSTab1.Tab = 3 ' hay que seleccionar device primero..
    Else
    SSTab1.Tab = gStartPage ' seleccion del usuario
    End If
    If gmonitor And gLineDev <> -1 Then
       lblMonitor.Visible = True
       lMonitor = False ' lo pone como si fuera apagado..
       cmdRepeat.Visible = False
       cmdOnOff_Click  ' inicia monitoreo
     End If
      If gMinimize And gLineDev <> -1 Then
       Me.WindowState = vbMinimized
     End If
      txtDialString = ""
    Me.Left = (Screen.Width - Me.Width) / 2
    Me.Top = (Screen.Height - Me.Height) / 2
      Exit Sub
  LocalErr:
    If err <> 0 Then
       lRtn = err
       cMsg = Error$
    End If
      MsgBox cMsg, vbCritical, "Form_Load Err[" & Hex(lRtn) & "]"
    TAPIShutdown
    MousePointer = vbNormal
    End ' final!!
End Sub
---------------------------------
Public Sub AddText(ctrl As Control, Data As String)
    Static intCounter As Integer
    intCounter = intCounter + 1
    Text1.Text = (intCounter) & ": " & Data & vbCrLf & ctrl
End Sub
---------------------------------
Public Sub TAPIPlaceCall(cdialtarget As String)
Dim x As Long
Dim encontro As Boolean
    encontro = False
    For x = 0 To ((DBGrid1.Rows) - 1)
     If DBGrid1.TextMatrix(x, 1) = gDialString Then
     encontro = True
     Else
     'nada por ahora
     End If
    Next x
    If encontro = False Then
       Dim iAns As Integer
       iAns = MsgBox("Agregar nuevo nombre a la Libreta de telefonos?", vbInformation + vbYesNo, "Agregar a
Libreta")
       If iAns = vbYes Then
          frmMain.Data1.Recordset.AddNew
```

```
        frmMain.Data1.Recordset.Fields("Nombre") = gName
        frmMain.Data1.Recordset.Fields("Telefono") = gDialString
        frmMain.Data1.Recordset.Fields("[Llamado]") = Now()
        frmMain.Data1.Recordset.Update
        frmMain.Data1.Refresh
      End If
    End If
    dpValue.DeviceNumber = gLineDev
    dpValue.Privilege = LINECALLPRIVILEGE_NONE
    dpValue.MediaMode = LINEMEDIAMODE_INTERACTIVEVOICE
    dpValue.DialableString = cdialtarget ' numero a llamar*** prueba para todos..
    lRtn = TAPIDial(dpValue)
    If lRtn < 0 Then
      cMsg = "TAPIDial - " & TapiErrMsg(lRtn)
      GoTo LocalErr
    End If
    Exit Sub
LocalErr:
    If err <> 0 Then
      lRtn = err
      cMsg = Error$
    End If
    MsgBox cMsg, vbCritical, "Dialing Err[" & Hex(lRtn) & "]"
End Sub
---------------------------------
Public Sub TAPIHangUp()
'DropCall
    a = lineDrop(hCall, "", 0)
    If a <> 0 Then
        ProcessTAPIError a
        AddText Me.Text1, TapiErrMsg(a) & "lineDrop -> UnSuccess"
        Else
        AddText Me.Text1, "lineDrop -> Success"
    End If
    a = lineDeallocateCall(hCall)
    If a <> 0 Then
        ProcessTAPIError a
        AddText Me.Text1, TapiErrMsg(a) & "lineDeallocateCall -> UnSuccess"
        Else
        AddText Me.Text1, "lineDeallocateCall -> Success"
    End If
    a = lineClose(hLine)
    If a <> 0 Then
        ProcessTAPIError a
        AddText Me.Text1, TapiErrMsg(a) & "lineClose -> UnSuccess"
        Else
        AddText Me.Text1, "lineClose -> Success"
    End If
End Sub
---------------------------------
Public Sub TAPIClearHandles()
    linehandler = 0 'HandleToCall = 0
    linehandler = 0
End Sub
---------------------------------
Public Sub TAPIShutdown()
    a = lineShutdown(hTAPI)
End Sub
---------------------------------
Public Sub ReadStartup()
    gMinimize = GetSetting(App.EXEName, "Startup", "MinimizeOnStart", 0)
    gmonitor = GetSetting(App.EXEName, "Startup", "Monitor", 0)
    gOutLog = GetSetting(App.EXEName, "Startup", "OutLog", 1)
    gOrigNumber = GetSetting(App.EXEName, "Startup", "OrigNumber", "")
    gStartPage = GetSetting(App.EXEName, "Startup", "StartPage", 0)
    gmonitor = 1
```

```
    chkMinimize = gMinimize
    chkMonitor = gmonitor
    chkOutLog = gOutLog
    If gOrigNumber = "" Then
       gOrigNumber = InputBox("Ingrese el Numero de su Telefono...", gOrigNumber)
       SaveSetting App.EXEName, "Startup", "OrigNumber", gOrigNumber
       Else
    End If
    txtOrigNumber.Text = gOrigNumber
    If gStartPage = 0 Then
       optStartup(0) = True
    Else
       optStartup(1) = True
    End If
End Sub
Public Sub InitDB()
    On Error Resume Next ' i'll handle this
    Open App.Path & "\tapi.mdb" For Input As 1
    If err = 53 Then
       MakeDB ' must create the database!
    Else
       Close #1 ' fine, just checking
    End If
    On Error GoTo LocalErr ' ok, pass them on...
    Data1.DatabaseName = App.Path & "\tapi.mdb"
    Data1.RecordSource = "SELECT * FROM TAPI"
    Data1.Refresh
    Data2.DatabaseName = App.Path & "\tapi.mdb"
    Data2.RecordSource = "TAPI"
    Data2.Refresh
    frmbases.Data3.DatabaseName = App.Path & "\tapi.mdb"
    frmbases.Data3.RecordSource = "SELECT Format(Llamado,'general date') AS Fecha, Nombre, Telefono,
Comentario FROM Log ORDER BY Nombre, Llamado"
    frmbases.Data3.Refresh
    frmbases.Data3b.DatabaseName = App.Path & "\tapi.mdb"
    frmbases.Data3b.RecordSource = "SELECT Format(Llamado,'general date') AS Fecha, Nombre, Telefono,
Comentario FROM Log2 ORDER BY Nombre, Llamado"
    frmbases.Data3b.Refresh
    Exit Sub
LocalErr:
    MsgBox Error$, vbCritical, "InitDB Err [" & CStr(err) & "]"
    End
End Sub
--------------------------------
Public Sub MakeDB()
    Dim cMakeTAPI As String
    Dim cMakeLOG As String
    Dim cMakeLOG2 As String
    Dim cDBName As String
    Dim ws As Workspace
    Dim db As Database
    cDBName = App.Path & "\TAPI.MDB"
    cMakeTAPI = "CREATE TABLE TAPI (Nombre TEXT(30), Telefono TEXT(20), [Llamado] DATE)"
    cMakeLOG = "CREATE TABLE LOG (Nombre TEXT(30), Llamado DATE, Telefono TEXT(20), Comentario
MEMO)"
    cMakeLOG2 = "CREATE TABLE LOG2 (Nombre TEXT(30), Llamado DATE, Telefono TEXT(20), Comentario
MEMO)"
    Set ws = Workspaces(0)
    Set db = ws.CreateDatabase(cDBName, dbLangGeneral)
    db.Execute cMakeTAPI, dbFailOnError
    db.Execute cMakeLOG, dbFailOnError
    db.Execute cMakeLOG2, dbFailOnError
    db.Close
    Set db = Nothing
    Set ws = Nothing
    Exit Sub
```


ESPOL 1958 ESCUELA SUPERIOR POLITECNICA DEL LITORAL CIB ESPOL

```
LocalErr:
   MsgBox Error$, vbCritical, "MakeDB Err [" & Str(err) & "]"
   End ' gotta stop here!
End Sub
```

ClsFile.cls

```
Public Function GetMessages() As Collection
Dim oColl As New Collection
Dim myString As String
   myString = Dir(App.Path & "\Messages\*.wav")
   Do While myString <> ""
      oColl.Add myString
      myString = Dir
   Loop
   Set GetMessages = oColl
End Function
```

Globals.bas

```
Option Explicit

Public nMessages As Long
Public hCall As Long
Public hTAPI As Long
Public lNumLines As Long
Public hLine As Long
Public lpLineDevCaps As linedevcaps
Public lMediaID As Long
Public m_TapiInit As Boolean
Public linehandler As Long
Public a As Long
Public strAddr As String
Public lpCallParams As Long
Public lLineID As Long
Public fSetup As Boolean
Public lpExtensionID As lineextensionid
Public lUnused As Long
Public lRtn As Long
Public cMsg As String
Public dpValue As DialParams
Public udtLineCall As LINECALLPARAMS
Public lngMsg As String
Public gDialString As String ' numero a llamar
Public gPlaceCall As Boolean ' ok a llamar
Public gName As String ' nombre a llamar
Public gName2 As String ' nombre a llamar2
Public nummen As Long ' posicion de mensajes
Public gLineDev As Integer ' line device
Public ultnombre As String ' ultimo llamado1
Public ultnumero As Long ' ultimo llamado2
Public gMinimize As Integer ' minimizar al inicio
Public gmonitor As Integer ' monitor al inicio
Public gOutLog As Integer ' llevar log de llamadas de salida (lo saque antes y lo he vuelto a meter..)
Public gOrigNumber As String ' calledID
Public gStartPage As Integer ' para inicio
Public ansnow As Boolean
Public CallIDName As String
Public CallIDNumber As String
Public sSecret As String   'Guarda los digitos del toll saver access
Public lTollSaver As Long   'contador de validacion del access code en el timer
Public flnTollSaver As Boolean 'flag para ver si hay mensajes en el telefono
Public lNegVer As Long     ' TAPI version
-----------------------------
Public Declare Sub OutputDebugString Lib "kernel32" Alias "OutputDebugStringA" (ByVal lpOutputString As String)
-------------------------------
Public Function LineCallBack(ByVal dwDevice As Long, ByVal dwMessage As Long, _
  ByVal dwInstance As Long, ByVal dwParam1 As Long, ByVal dwParam2 As Long, _
  ByVal dwParam3 As Long) As Long
  Select Case dwMessage
    Case TapiEvent.LINE_ADDRESSSTATE
      DebugString 4, "LINE_ADDRESSSTATE"
    Case TapiEvent.LINE_CALLINFO
      DebugString 3, "LINE_CALLINFO"
      If dwParam1 = LINECALLINFOSTATE_CALLERID Then
        DebugString 4, "LINECALLINFOSTATE_CALLERID"
        GetCallerInfo dwDevice
      Else
        DebugString 5, "LINE_CALLINFO -> " & CStr(dwParam1)
      End If
    Case TapiEvent.LINE_CALLSTATE
      DebugString 3, "LINE_CALLSTATE"
      LineCallStateProc dwDevice, dwInstance, dwParam1, dwParam2, dwParam3
```

```
      Case TapiEvent.LINE_CLOSE
         DebugString 4, "LINE_CLOSE"
      Case TapiEvent.LINE_CREATE
         DebugString 4, "LINE_CREATE:"
      Case TapiEvent.LINE_DEVSPECIFIC
         DebugString 4, "LINE_DEVSPECIFIC"
      Case TapiEvent.LINE_DEVSPECIFICFEATURE
         DebugString 4, "LINE_DEVSPECIFICFEATURE"
      Case TapiEvent.LINE_GATHERDIGITS
         DebugString 4, "LINE_GATHERDIGITS"
      Case TapiEvent.LINE_GENERATE
         DebugString 3, "LINE_GENERATE"
         RecordMessage
      Case TapiEvent.LINE_LINEDEVSTATE
         DebugString 3, "LINE_LINEDEVSTATE"
         LineDevStateProc dwDevice, dwInstance, dwParam1, dwParam2, dwParam3
      Case TapiEvent.LINE_MONITORDIGITS
         DebugString 3, "LINE_MONITORDIGITS -> " & Chr(LoWord(dwParam1))
         sSecret = sSecret & CStr(Chr(LoWord(dwParam1)))
         lTollSaver = 0  'Resetea el toll saver counter
      Case TapiEvent.LINE_MONITORMEDIA
         DebugString 4, "LINE_MONITORMEDIA"
      Case TapiEvent.LINE_MONITORTONE
         DebugString 4, "LINE_MONITORTONE"
      Case TapiEvent.LINE_REPLY
          DebugString 3, "LINE_REPLY -> " & CStr(dwParam2)
      Case TapiEvent.LINE_REQUEST
         DebugString 4, "LINE_REQUEST"
      Case TapiEvent.PHONE_BUTTON
         DebugString 4, "PHONE_BUTTON"
      Case TapiEvent.PHONE_CLOSE
         DebugString 4, "PHONE_CLOSE"
      Case TapiEvent.PHONE_CREATE
         DebugString 4, "PHONE_CREATE"
      Case TapiEvent.PHONE_DEVSPECIFIC
         DebugString 4, "PHONE_DEVSPECIFIC"
      Case TapiEvent.PHONE_REPLY
         DebugString 4, "PHONE_REPLY"
      Case TapiEvent.PHONE_STATE
         DebugString 4, "PHONE_STATE"
      Case Else
         DebugString 2, "Unknown TAPI Event: " & CStr(dwMessage)
   End Select
   frmMain.AddText frmMain.Text1, "Mensajes Tapi: " & LineMsg(dwMessage)
   frmMain.AddText frmMain.Text1, " Param1:" & CStr(dwParam1) & " Param2:" & CStr(dwParam2) & " Param3:" &
CStr(dwParam3)
End Function
--------------------------------
Public Function LineMsg(lngMsg) As String
   Dim strTemp As String
   Select Case lngMsg
      Case ls = LINE_ADDRESSSTATE ' 0&
         strTemp = "LINE_ADDRESSSTATE"
      Case ls = LINE_CALLINFO ' 1&
         strTemp = "LINE_CALLINFO"
      Case ls = LINE_CALLSTATE ' 2&
         strTemp = "LINE_CALLSTATE"
      Case ls = LINE_CLOSE ' 3&
         strTemp = "LINE_CLOSE"
      Case ls = LINE_DEVSPECIFIC ' 4&
         strTemp = "LINE_DEVSPECIFIC"
      Case ls = LINE_DEVSPECIFICFEATURE ' 5&
         strTemp = "LINE_DEVSPECIFICFEATURE"
      Case ls = LINE_GATHERDIGITS ' 6&
         strTemp = "LINE_GATHERDIGITS"
      Case ls = LINE_GENERATE ' 7&
```

```
        strTemp = "LINE_GENERATE"
      Case ls = LINE_LINEDEVSTATE ' 8&
        strTemp = "LINE_LINEDEVSTATE"
      Case ls = LINE_MONITORDIGITS ' 9&
        strTemp = "LINE_MONITORDIGITS"
      Case ls = LINE_MONITORMEDIA ' 10&
        strTemp = "LINE_MONITORMEDIA"
      Case ls = LINE_MONITORTONE ' 11&
        strTemp = "LINE_MONITORTONE"
      Case ls = LINE_REPLY ' 12&
        strTemp = "LINE_REPLY"
      Case ls = LINE_REQUEST ' 13&
        strTemp = "LINE_REQUEST"
      Case ls = PHONE_BUTTON ' 14&
        strTemp = "PHONE_BUTTON"
      Case ls = PHONE_CLOSE ' 15&
        strTemp = "PHONE_CLOSE"
      Case ls = PHONE_DEVSPECIFIC ' 16&
        strTemp = "PHONE_DEVSPECIFIC"
      Case ls = PHONE_REPLY ' 17&
        strTemp = "PHONE_REPLY"
      Case ls = PHONE_STATE ' 18&
        strTemp = "PHONE_STATE"
      Case Else
        strTemp = "Unknown Message"
    End Select
    LineMsg = strTemp & " [" & CStr(lngMsg) & "]"
End Function
---------------------------------
Public Function GetCallerInfo(hCall As Long) As Long
Dim lpCallInfo As lineCallInfo
Dim nErr As Long
Dim sCallerID As String
Dim sCallerName As String
Dim lStart As Long
Dim lLength As Long
Dim lLoop As Long
On Error GoTo EH
        CallIDName = "Entrando llamada... en procèso.."
        frmMain.lblName.Caption = CallIDName
        CallIDNumber = "Entrando llamada... en proceso.."
        frmMain.lblNumber.Caption = CallIDNumber
    lpCallInfo.dwTotalSize = Len(lpCallInfo)
    nErr = lineGetCallInfo(hCall, lpCallInfo)
    If nErr <> 0 Then
      ProcessTAPIError nErr
      GetCallerInfo = -1
      Exit Function
    End If
    If (lpCallInfo.dwCallerIDFlags And LINECALLPARTYID_ADDRESS) <> False Then
    lStart = Len(lpCallInfo) - UBound(lpCallInfo.bBytes())
    lStart = lpCallInfo.dwCallerIDOffset - lStart + 1
    lLength = lpCallInfo.dwCallerIDSize
    For lLoop = 0 To lLength
      If lpCallInfo.bBytes(lStart + lLoop) = 0 Then Exit For
      sCallerID = sCallerID & CStr(Chr(lpCallInfo.bBytes(lStart + lLoop)))
    Next
    CallIDNumber = sCallerID
    Else
      CallIDNumber = "Desconocido"
    End If
    If (lpCallInfo.dwCallerIDFlags And LINECALLPARTYID_NAME) <> False Then
    lStart = Len(lpCallInfo) - UBound(lpCallInfo.bBytes())
    lStart = lpCallInfo.dwCallerIDNameOffset - lStart + 1
    lLength = lpCallInfo.dwCallerIDNameSize
    For lLoop = 0 To lLength
```

```
      If lpCallInfo.bBytes(lStart + lLoop) = 0 Then Exit For
      sCallerName = sCallerName & CStr(Chr(lpCallInfo.bBytes(lStart + lLoop)))
    Next
    CallIDName = sCallerName
    Else
    CallIDName = "Desconocido"
    Dim x As Long
    For x = 0 To ((frmMain.DBGrid1.Rows) - 1)
     If frmMain.DBGrid1.TextMatrix(x, 1) = sCallerID Then
     CallIDName = frmMain.DBGrid1.TextMatrix(x, 0)
    Else
      'nada por ahora
    End If
    Next
    End If
    DebugString 5, "CallerID:"
    DebugString 4, sCallerID & " : " & sCallerName
    GetCallerInfo = 0
    frmMain.lblName.Caption = CallIDName
    frmMain.lblNumber.Caption = CallIDNumber
    With frmbases.Data3b.Recordset
      .AddNew
      .Fields("Nombre") = frmMain.lblName.Caption
      .Fields("Llamado") = Now()
      .Fields("Telefono") = frmMain.lblNumber.Caption
      .Fields("Comentario") = "llamada entrante "
      .Update
    End With
    frmbases.Data3b.Refresh
    frmbases.MSFlexGrid2.Refresh
    frmMain.lblMsgCount2.Caption = ((frmbases.MSFlexGrid2.Rows) - 1)
    frmMain.Data1.Recordset.FindFirst "Nombre='" & frmMain.lblName.Caption & "'"
    If frmMain.Data1.Recordset.NoMatch = False Then
      With frmMain.Data1.Recordset
        .Edit
        .Fields("[Llamado]") = Now()
        .Update
      End With
    End If
    frmMain.Data1.Refresh
    Exit Function
EH:
Debug.Print err.Number & " " & err.Description
GetCallerInfo = 1
End Function
---------------------------------
Public Sub LineDevStateProc(ByVal dwDevice As Long, ByVal dwInstance As Long, _
        ByVal dwParam1 As Long, ByVal dwParam2 As Long, _
        ByVal dwParam3 As Long)
    Select Case dwParam1
      Case LINEDEVSTATE_OTHER:
        DebugString 5, "LINEDEVSTATE_OTHER:"
      Case LINEDEVSTATE_RINGING:  'el unico mensaje LineDevState que usamos..
        DebugString 3, "LINEDEVSTATE_RINGING:"
        DebugString 3, "Conteo de Rings = " & CStr(dwParam3)
        If ansnow = True Then GoTo salir
        If frmMain.lblMsgCount > 0 Then
          frmMain.borrar.Visible = True
          frmMain.borrart.Visible = True
          If dwParam3 >= dlgSetup.txtTollSaver Then Answer
        Else
          If dwParam3 >= dlgSetup.txtRTA Then Answer
        End If
salir:
      Case LINEDEVSTATE_CONNECTED:
        DebugString 5, "LINEDEVSTATE_CONNECTED:"
```

```
    Case LINEDEVSTATE_DISCONNECTED:
      DebugString 5, "LINEDEVSTATE_DISCONNECTED:"
    Case LINEDEVSTATE_MSGWAITON:
      DebugString 5, "LINEDEVSTATE_MSGWAITON:"
    Case LINEDEVSTATE_MSGWAITOFF:
      DebugString 5, "LINEDEVSTATE_MSGWAITOFF:"
    Case LINEDEVSTATE_INSERVICE:
      DebugString 5, "LINEDEVSTATE_INSERVICE:"
    Case LINEDEVSTATE_OUTOFSERVICE:
      DebugString 5, "LINEDEVSTATE_OUTOFSERVICE:"
    Case LINEDEVSTATE_MAINTENANCE:
      DebugString 5, "LINEDEVSTATE_MAINTENANCE:"
    Case LINEDEVSTATE_OPEN:
      DebugString 5, "LINEDEVSTATE_OPEN:"
    Case LINEDEVSTATE_CLOSE:
      DebugString 5, "LINEDEVSTATE_CLOSE:"
    Case LINEDEVSTATE_NUMCALLS:
      DebugString 5, "LINEDEVSTATE_NUMCALLS:"
    Case LINEDEVSTATE_NUMCOMPLETIONS:
      DebugString 5, "LINEDEVSTATE_NUMCOMPLETIONS:"
    Case LINEDEVSTATE_TERMINALS:
      DebugString 5, "LINEDEVSTATE_TERMINALS:"
    Case LINEDEVSTATE_ROAMMODE:
      DebugString 5, "LINEDEVSTATE_ROAMMODE:"
    Case LINEDEVSTATE_BATTERY:
      DebugString 5, "LINEDEVSTATE_BATTERY:"
    Case LINEDEVSTATE_SIGNAL:
      DebugString 5, "LINEDEVSTATE_SIGNAL:"
    Case LINEDEVSTATE_DEVSPECIFIC:
      DebugString 5, "LINEDEVSTATE_DEVSPECIFIC:"
    Case LINEDEVSTATE_REINIT:
      DebugString 5, "LINEDEVSTATE_REINIT:"
    Case LINEDEVSTATE_LOCK:
      DebugString 5, "LINEDEVSTATE_LOCK:"
    Case LINEDEVSTATE_CAPSCHANGE:
      DebugString 5, "LINEDEVSTATE_CAPSCHANGE:"
    Case LINEDEVSTATE_CONFIGCHANGE:
      DebugString 5, "LINEDEVSTATE_CONFIGCHANGE:"
    Case LINEDEVSTATE_TRANSLATECHANGE:
      DebugString 5, "LINEDEVSTATE_TRANSLATECHANGE:"
    Case LINEDEVSTATE_COMPLCANCEL:
      DebugString 5, "LINEDEVSTATE_COMPLCANCEL:"
    Case LINEDEVSTATE_REMOVED:
      DebugString 5, "LINEDEVSTATE_REMOVED:"
    Case Else:
      DebugString 2, "LINEDEVSTATE_UNKNOWN:"
  End Select
End Sub
---------------------------------
Public Sub LineCallStateProc(ByVal dwDevice As Long, ByVal dwInstance As Long, _
        ByVal dwParam1 As Long, ByVal dwParam2 As Long, ByVal dwParam3 As Long)
  Select Case dwParam1
    Case LINECALLSTATE_IDLE:
        DebugString 3, "LINECALLSTATE_IDLE:" ' me colgaron!!!
        ansnow = False
        frmMain.cmdButtons_Click 2 ' proceso de cuelgo local
        frmMain.TAPIClearHandles
    Case LINECALLSTATE_OFFERING:
      DebugString 3, "LINECALLSTATE_OFFERING:"
      hCall = dwDevice 'actualizo el handler
      Beep ' para indicar al usuario que llego una llamada!!
      Beep
      Beep
      frmMain.WindowState = vbNormal ' maximiza, si es necesario
      frmMain.cmdButtons(1).Visible = False
      frmMain.Command1.Visible = True
```

```
            frmMain.Label5.Visible = True
        Case LINECALLSTATE_ACCEPTED:
            DebugString 4, "LINECALLSTATE_ACCEPTED:"
        Case LINECALLSTATE_DIALTONE:
            DebugString 5, "LINECALLSTATE_DIALTONE:"
        Case LINECALLSTATE_DIALING:
            DebugString 5, "LINECALLSTATE_DIALING:"
        Case LINECALLSTATE_RINGBACK:
            DebugString 5, "LINECALLSTATE_RINGBACK:"
        Case LINECALLSTATE_BUSY:
            DebugString 5, "LINECALLSTATE_BUSY:"
        Case LINECALLSTATE_SPECIALINFO:
            DebugString 4, "LINECALLSTATE_SPECIALINFO:"
        Case LINECALLSTATE_CONNECTED:
            DebugString 4, "LINECALLSTATE_CONNECTED:"
            fInTollSaver = False
            If ansnow = True Then GoTo salir
            PlayGreeting
salir:
        Case LINECALLSTATE_PROCEEDING:
            DebugString 5, "LINECALLSTATE_PROCEEDING:"
        Case LINECALLSTATE_ONHOLD:
            DebugString 5, "LINECALLSTATE_ONHOLD:"
        Case LINECALLSTATE_CONFERENCED:
            DebugString 5, "LINECALLSTATE_CONFERENCED:"
        Case LINECALLSTATE_ONHOLDPENDCONF:
            DebugString 5, "LINECALLSTATE_ONHOLDPENDCONF:"
        Case LINECALLSTATE_ONHOLDPENDTRANSFER:
            DebugString 5, "LINECALLSTATE_ONHOLDPENDTRANSFER:"
        Case LINECALLSTATE_DISCONNECTED:
            DebugString 4, "LINECALLSTATE_DISCONNECTED:"
            ansnow = False
            Call waveInReset(h_wavein)
            DebugString 5, "m_BPlayRec -> " & CStr(m_BPlayRec)
            If fInTollSaver = True Then
                frmMain.StopPlaying
                DropCall
            End If
            frmMain.cmdButtons_Click (2)
        Case LINECALLSTATE_UNKNOWN:
            DebugString 4, "LINECALLSTATE_UNKNOWN:"
    End Select
End Sub
---------------------------------
Public Sub DebugString(nSeverity As Long, Message As String)
frmMain.AddText frmMain.Text1, "Tapi Callback: " & Message
    OutputDebugString Message & vbCrLf
End Sub
---------------------------------
Function LoWord(ByVal dw As Long) As Integer
    If dw And &H8000& Then
        LoWord = dw Or &HFFFF0000
    Else
        LoWord = dw And &HFFFF
    End If
End Function
---------------------------------
Function LShiftWord(w As Long, c As Integer) As Long
    LShiftWord = w * (2 ^ c)
End Function
---------------------------------
Public Sub Answer()
On Error Resume Next
Dim nError As Long
    nError = lineAnswer(hCall, "", 0)
    If nError < 0 Then ProcessTAPIError nError
```

```
    frmMain.Command1.Visible = False
    frmMain.cmdButtons(1).Visible = True
    frmMain.Label5.Visible = False
    If ansnow = False Then MonitorDigits
End Sub
---------------------------------
Public Sub GetLineID(sWave As String)
On Error Resume Next
Dim nError As Long
Dim sTemp As String
Dim oVar As varString
    oVar.dwTotalSize = Len(oVar)
    nError = lineGetID(hLine, 0, hCall, LINECALLSELECT_CALL, oVar, sWave)
    If nError <> 0 Then
        ProcessTAPIError nError
    Else
        If oVar.dwStringOffset = 0 Then 'No hay nada
            lMediaID = -1
            Exit Sub
        End If
        sTemp = Trim(Left(oVar.bBytes(0), oVar.dwStringSize))
        lMediaID = sTemp
        DebugString 5, "LineID: " & CStr(lMediaID)
    End If
End Sub
---------------------------------
Private Sub PlayGreeting()
If ansnow = True Then GoTo salir
On Error GoTo EH
    GetLineID "wave/out"
    frmMain.lblName.Caption = CallIDName
    frmMain.lblNumber.Caption = CallIDNumber
    frmMain.LblTime.Caption = Now
    If m_BPlayRec <> False Then
        If (fPlaying = False) Then
          ' -1 especifica el wave mapper
          LoadFile App.Path & "\" & "Greeting.wav"
          Play lMediaID
          m_BPlayRec = True
        End If
    End If
    Exit Sub
EH:
DebugString 0, CStr(err.Number) & " : " & err.Description
salir:
End Sub
---------------------------------
Private Sub RecordMessage()
If ansnow = True Then GoTo salir
On Error GoTo EH
    GetLineID "wave/in"
    nMessages = nMessages + 1
    RecStart dlgSetup.txtMsgLen, lMediaID, CallIDName & "~" & CallIDNumber & "~" _
        & TimeAsString & ".wav"
    Exit Sub
EH:
DebugString 0, CStr(err.Number) & " : " & err.Description
salir:
End Sub
---------------------------------
Private Function TimeAsString() As String
Dim lPos As Long
Dim strNow As String
    strNow = Now
    lPos = InStr(1, strNow, "/")
    Do While lPos > 0
```

```
    strNow = Mid(strNow, 1, lPos - 1) & "-" & Mid(strNow, lPos + 1, Len(strNow))
    lPos = InStr(1, strNow, "/")
    Loop
    lPos = InStr(1, strNow, ":")
    Do While lPos > 0
    strNow = Mid(strNow, 1, lPos - 1) & "-" & Mid(strNow, lPos + 1, Len(strNow))
    lPos = InStr(1, strNow, ":")
    Loop
    TimeAsString = strNow
End Function
---------------------------------
Private Sub MonitorDigits()
Dim nError As Long
    nError = lineMonitorDigits(hCall, LINEDIGITMODE_DTMF)
    If nError <> 0 Then
        ProcessTAPIError nError
    Else
        DebugString 5, "lineMonitorDigits -> Success"
    End If
        End Sub
---------------------------------
Public Sub SendDigit()
Dim nError As Long
    nError = lineGenerateDigits(hCall, LINEDIGITMODE_DTMF, _
                        "#", 0)
    If nError <> 0 Then ProcessTAPIError nError
End Sub
---------------------------------
Public Sub DropCall()
    Call lineDrop(hCall, "", 0)
    Call lineDeallocateCall(hCall)
    ansnow = False
    frmMain.Command1.Visible = False
    frmMain.cmdButtons(1).Visible = True
End Sub
---------------------------------
Public Sub SetCallParams(lMode As Long)
    Dim cp As LINECALLPARAMS
    Dim cpx As String
    cp.dwMinRate = 9.6
    cp.dwMaxRate = 28.8
    cp.dwMediaMode = lMode
    cp.dwCallParamFlags = 0
    cp.dwAddressMode = lMode
    cp.DialParams.dwDialPause = 0
    cp.DialParams.dwDialSpeed = 0
    cp.DialParams.dwDigitDuration = 0
    cp.DialParams.dwWaitForDialtone = 0
    cp.dwOrigAddressSize = Len(Trim(gOrigNumber))
    cp.dwOrigAddressOffset = 0
    cp.dwCalledPartySize = 0
    cp.dwCommentSize = 0
    cp.dwUserUserInfoSize = 0
    cp.dwHighLevelCompSize = 0
    cp.dwLowLevelCompSize = 0
    cp.dwDevSpecificSize = 0
    cpx = Trim(gOrigNumber)
    cp.dwTotalSize = Len(cp) + Len(cpx)
End Sub
---------------------------------
Public Function TAPIDial(dpType As DialParams) As Long
    Dim a As Long
    Dim strAddr As String
    frmMain.AddText frmMain.Text1, m_TapiInit & " = m_TapiInit"

    If m_TapiInit = False Then
```

```
    frmMain.myInit False 'si tapi no esta iniciado, lo inicio ahora...
    Else
    'no pasa nada
  End If
    lLineID = GetSetting("VB-TAPI", "Settings", "DeviceID", "0")
  strAddr = dpType.DialableString
  frmMain.AddText frmMain.Text1, linehandler & " = linehandler"
  frmMain.AddText frmMain.Text1, "Numero: " & strAddr
  frmMain.AddText frmMain.Text1, hLine & " = hLine"
  frmMain.AddText frmMain.Text1, hCall & " = hCall"
  frmMain.AddText frmMain.Text1, lpCallParams & " = lpCallParams"
  If Trim(strAddr) = "" Then GoTo salir
  a = lineMakeCall(hLine, hCall, strAddr, 1, lpCallParams) 'verdadero
  If a < 0 Then
    frmMain.AddText frmMain.Text1, TapiErrMsg(a) & " - lineMakeCall"
  Else
    frmMain.AddText frmMain.Text1, "Call placed! (" & strAddr & ") - lineMakeCall"
  End If
salir:
End Function
```

DlGreeting.frm

```
Private Sub cmdPlay_Click()
On Error GoTo EH
cmdRecord.Enabled = False
   grab.Visible = True
   grab2.Visible = False
   If m_BPlayRec <> False Then
      If (fPlaying = False) Then
       ' -1 especifica el wave mapper
       LoadFile App.Path & "\" & "Greeting.wav"
       Play -1
       m_BPlayRec = True
      End If
   End If
   Exit Sub
EH:
MsgBox CStr(err.Number) & " : " & err.Description
End Sub
---------------------------------
Private Sub cmdRecord_Click()
On Error GoTo EH
   cmdPlay.Enabled = False
   grab.Visible = True
   grab2.Visible = False
   If fPlaying = False Then
      RecStart 20, -1, "Greeting.wav"
      m_BPlayRec = False
   End If
   Exit Sub
EH:
   MsgBox CStr(err.Number) & " : " & err.Description
End Sub
---------------------------------
Private Sub cmdStop_Click()
Dim nErr As Long
   cmdPlay.Enabled = True
   grab.Visible = False
   grab2.Visible = True
   cmdRecord.Enabled = True
   If m_BPlayRec = False Then
      nErr = waveInReset(h_wavein)
   Else
      nErr = waveOutReset(hWaveOut)
   End If
   'If nErr <> 0 Then MsgBox "Error Reseteando Wave Device"
End Sub
---------------------------------
Private Sub Form_Load()
   Me.Top = frmMain.Top + ((frmMain.Height - Me.Height) / 2)
   Me.Left = frmMain.Left + ((frmMain.Width - Me.Width) / 2)

   If m_TapiInit = True Then frmMain.myInit (False)

End Sub
---------------------------------
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   If frmMain.Timer1.Enabled = True Then Cancel = 1 'Aun toca mensaje-greeting
End Sub
---------------------------------
Private Sub Form_Unload(Cancel As Integer)

If m_TapiInit = False Then frmMain.myInit (False)

End Sub
```

DlgSetup.frm

```
Private Sub Form_Load()
   Me.Top = frmMain.Top + ((frmMain.Height - Me.Height) / 2)
   Me.Left = frmMain.Left + ((frmMain.Width - Me.Width) / 2)
   txtMsgLen = GetSetting("VB-TAPI", "Settings", "MaxMessage", "60")
   txtRTA = GetSetting("VB-TAPI", "Settings", "NumRings", "5")
   txtTollSaver = GetSetting("VB-TAPI", "Settings", "TollSaver", "3")
   txtSecret = GetSetting("VB-TAPI", "Settings", "Secret", "123")
End Sub
---------------------------------
Private Sub Form_Resize()
Dim lpLineDevCaps As linedevcaps
Dim lLoop As Long
Dim lUnused As Long
Dim nError As Long
On Error GoTo EH

   Screen.MousePointer = vbHourglass
   frmMain.myInit True

   For lLoop = 0 To lNumLines
   lpLineDevCaps.dwTotalSize = Len(lpLineDevCaps)
   nError = lineGetDevCaps(hTAPI, lLoop, lNegVer, lUnused, lpLineDevCaps)
   If nError <> 0 Then
      'salto:
   Else
      Dim sTemp As String
      Dim lTemp As Long
      Dim lStart As Long
      lStart = Len(lpLineDevCaps) - UBound(lpLineDevCaps.bBytes())
      lStart = lpLineDevCaps.dwLineNameOffset - lStart + 1
      For lTemp = 0 To lpLineDevCaps.dwLineNameSize
         If lpLineDevCaps.bBytes(lStart + lTemp) = 0 Then Exit For
         sTemp = sTemp & CStr(Chr(lpLineDevCaps.bBytes(lStart + lTemp)))
      Next
   End If
   DebugString 4, sTemp
   If sTemp = "" Then
      cmbDevice.AddItem "Unknown"
   Else
      cmbDevice.AddItem sTemp
   End If
   sTemp = ""
   Next
   Screen.MousePointer = vbNormal
   cmbDevice.ListIndex = GetSetting("VB-TAPI", "Settings", "DeviceID", "0")
   Exit Sub
EH:
Screen.MousePointer = vbNormal
MsgBox err.Number & " : " & err.Description
End Sub
---------------------------------
Private Sub Form_Unload(Cancel As Integer)
   On Error Resume Next
   frmMain.myInit True
   frmMain.myInit False
   Screen.MousePointer = vbNormal
End Sub
---------------------------------
Private Sub OKButton_Click()
SaveSetting "VB-TAPI", "Settings", "MaxMessage", txtMsgLen
SaveSetting "VB-TAPI", "Settings", "NumRings", txtRTA
SaveSetting "VB-TAPI", "Settings", "TollSaver", txtTollSaver
```

```
SaveSetting "VB-TAPI", "Settings", "Secret", txtSecret
SaveSetting "VB-TAPI", "Settings", "DeviceID", CStr(cmbDevice.ListIndex)
Unload Me
End Sub
---------------------------------
Private Sub CancelButton_Click()
   Unload Me
End Sub
---------------------------------
Private Sub txtMsgLen_Change()
   If Not IsNumeric(txtMsgLen.Text) Then
      Beep
      txtMsgLen = GetSetting("VB-TAPI", "Settings", "MaxMessage", "60")
   End If
End Sub
---------------------------------
Private Sub txtRTA_Change()
   If Not IsNumeric(txtRTA.Text) Then
      Beep
      txtRTA = GetSetting("VB-TAPI", "Settings", "NumRings", "5")
   End If
End Sub
---------------------------------
Private Sub txtSecret_Change()
   If Not IsNumeric(txtSecret.Text) Then
      Beep
      txtSecret = GetSetting("VB-TAPI", "Settings", "Secret", "123")
   End If
End Sub
---------------------------------
Private Sub txtTollSaver_Change()
   If Not IsNumeric(txtTollSaver.Text) Then
      Beep
      txtTollSaver = GetSetting("VB-TAPI", "Settings", "TollSaver", "3")
   End If
End Sub
```

FrmBases.frm

```
Private Sub cmdLog_Click(Index As Integer)
   Dim iAns As Integer ' respuesta del usuario
   Select Case Index
      Case 0 ' limpia historial
         ClearLog
      Case 1 ' actualiza
         Data3.Refresh
      Case 2 ' exporta a archivo de texto
         ExportLog
   End Select
End Sub
---------------------------------
Public Sub ClearLog2()
   On Error GoTo LocalErr
   Dim iAns As Integer
   iAns = MsgBox("Listo para limpiar historial..", vbInformation + vbYesNo, "Limpiar Historial")
   If iAns = vbYes Then
      MousePointer = vbHourglass ' cambia puntyero a ocupado..de puro cache..je!
      Data3b.Recordset.MoveFirst ' comienza por arriba
      Do Until Data3b.Recordset.EOF ' sigue hasta que limpia todo..
         Data3b.Recordset.Delete ' limpio uno y..
         Data3b.Recordset.MoveNext ' voy al siguiente..
      Loop ' de nuevo para arriba..
      MousePointer = vbNormal ' digo que ya termine y cambio cursor
      MsgBox "Historial Limpiado!!!", vbInformation
   End If
   Data3b.Refresh
   Exit Sub
LocalErr:
   MsgBox Error$, vbCritical, "Historial Err [" & CStr(err) & "]"
End Sub
---------------------------------
Public Sub ExportLog2()
   Dim cFldDelim As String
   Dim cLine As String
   Dim cFileName As String
   Dim cQuote As String
   cFldDelim = Chr(44)
   cQuote = Chr(34)
   cFileName = App.Path & "\" & App.EXEName & ".log"
   Open cFileName For Output As 1
   cLine = "Fecha,Nombre,Numero,Comentario"
   Print #1, cLine
   Data3b.Recordset.MoveFirst
   Do Until Data3b.Recordset.EOF
      With Data3b.Recordset
         cLine = cQuote & Trim(.Fields("DateCalled")) & cQuote & cFldDelim
         cLine = cLine & cQuote & Trim(.Fields("Name")) & cQuote & cFldDelim
         cLine = cLine & cQuote & Trim(.Fields("NumberCalled")) & cQuote & cFldDelim
         cLine = cLine & cQuote & Trim(.Fields("Comment")) & cQuote
         Print #1, cLine
         .MoveNext
      End With
   Loop
   Close #1
   MsgBox "Exportando Historial a [" & cFileName & "]", vbInformation, "... Historial exportado!"
End Sub
---------------------------------
Private Sub cmdLog1_Click(Index As Integer)
   Dim iAns As Integer ' respuesta del usuario
   Select Case Index
      Case 0 ' limpia historial
```

```
        ClearLog2
      Case 1 ' actualiza
        Data3b.Refresh
      Case 2 ' exporta a archivo de texto
        ExportLog2
  End Select
End Sub
---------------------------------
Private Sub Form_Load()
  Data3.DatabaseName = App.Path & "\tapi.mdb"
  Data3.RecordSource = "Log"
  Data3.Refresh
  Data3b.DatabaseName = App.Path & "\tapi.mdb"
  Data3b.RecordSource = "Log2"
  Data3b.Refresh
End Sub
---------------------------------
Public Sub ClearLog()
  On Error GoTo LocalErr
  Dim iAns As Integer
  iAns = MsgBox("Listo para limpiar historial..", vbInformation + vbYesNo, "Limpiar Historial")
  If iAns = vbYes Then
    MousePointer = vbHourglass ' cambia puntyero a ocupado..de puro cache..je!
    Data3.Recordset.MoveFirst ' comienza por arriba
    Do Until Data3.Recordset.EOF ' sigue hasta que limpia todo..
      Data3.Recordset.Delete ' limpio uno y..
      Data3.Recordset.MoveNext ' voy al siguiente..
    Loop ' de nuevo para arriba..
    MousePointer = vbNormal ' digo que ya termine y cambio cursor
    MsgBox "Historial Limpiado!!!", vbInformation
  End If
  Data3.Refresh
  Exit Sub
LocalErr:
  MsgBox Error$, vbCritical, "Historial Err [" & CStr(err) & "]"
End Sub
---------------------------------
Public Sub ExportLog()
  Dim cFldDelim As String
  Dim cLine As String
  Dim cFileName As String
  Dim cQuote As String
  cFldDelim = Chr(44)
  cQuote = Chr(34)
  cFileName = App.Path & "\" & App.EXEName & ".log"
  Open cFileName For Output As 1
  cLine = "Fecha,Nombre,Numero,Comentario"
  Print #1, cLine
  Data3.Recordset.MoveFirst
  Do Until Data3.Recordset.EOF
    With Data3.Recordset
      cLine = cQuote & Trim(.Fields("DateCalled")) & cQuote & cFldDelim
      cLine = cLine & cQuote & Trim(.Fields("Name")) & cQuote & cFldDelim
      cLine = cLine & cQuote & Trim(.Fields("NumberCalled")) & cQuote & cFldDelim
      cLine = cLine & cQuote & Trim(.Fields("Comment")) & cQuote
      Print #1, cLine
      .MoveNext
    End With
  Loop
  Close #1
  MsgBox "Exportando Historial a [" & cFileName & "]", vbInformation, "... Historial exportado!"
End Sub
```

FrmCall.frm

```
Private Sub cmdCall_Click(Index As Integer)
   Select Case Index
      Case 0 ' llamar
         gName = lblName.Text
         gPlaceCall = True ' ok a llamada
         UpdateDB ' actualiza base de datos
      Case 1 ' cancela
         gPlaceCall = False ' lo salto
   End Select
   Unload Me
End Sub
---------------------------------
Private Sub Form_Load()
   Dim x As Long
Dim encontro As Boolean
   encontro = False
   For x = 0 To ((frmMain.DBGrid1.Rows) - 1)
    If frmMain.DBGrid1.TextMatrix(x, 1) = gDialString Then
    encontro = True
    gName = frmMain.DBGrid1.TextMatrix(x, 0)
    Else
    'nada por ahora
    End If
   Next x
   lblDialString = Trim(gDialString)
   lblName.Text = Trim(gName)
   txtNotes.Text = ""
   ultnombre = lblName.Text
   ultnumero = lblDialString
   Me.Left = (Screen.Width - Me.Width) / 2
   Me.Top = (Screen.Height - Me.Height) / 2
   Data1.DatabaseName = App.Path & "\tapi.mdb"
   Data1.RecordSource = "Log"
   Data1.Refresh
End Sub
---------------------------------
Public Sub UpdateDB()
   With frmCall.Data1.Recordset
      .AddNew
      .Fields("Nombre") = lblName.Text
      .Fields("Llamado") = Now()
      .Fields("Telefono") = lblDialString
      .Fields("Comentario") = txtNotes.Text & " "
      .Update
   End With
   frmCall.Data1.Refresh
   frmMain.Data1.Recordset.FindFirst "Nombre='" & lblName.Text & "'"
   If frmMain.Data1.Recordset.NoMatch = False Then
      With frmMain.Data1.Recordset
         .Edit
         .Fields("[Llamado]") = Now()
         .Update
      End With
   End If
   frmMain.Data1.Refresh
End Sub
```

FrmData.frm

```
Private Sub Command1_Click()
Dim x As Long
Dim encontro As Boolean
Dim encontro2 As Boolean
   If txtName.Text = "" Then
   gName = MsgBox("Ingrese un Nombre por favor...", vbOK)
   Exit Sub
   End If
   If txtPhoneNumber.Text = "" Then
   gName = MsgBox("Ingrese un Numero por favor...", vbOK)
   Exit Sub
   End If
   encontro = False
   encontro2 = False
   For x = 0 To ((frmMain.DBGrid1.Rows) - 1)
    If frmMain.DBGrid1.TextMatrix(x, 1) = txtPhoneNumber.Text Then
    encontro = True
    Else
    'nada por ahora
    End If
   Next x
   For x = 0 To ((frmMain.DBGrid1.Rows) - 1)
    If frmMain.DBGrid1.TextMatrix(x, 0) = txtName.Text Then
    encontro2 = True
    Else
    'nada por ahora
    End If
   Next x
If (encontro = True) And (encontro2 = True) Then
   GoTo mevoy
   Else
   'nada
   End If
   If (encontro = True) Then
      gName = MsgBox("El telefono ya existe..agregar otro usuario con este numero?", vbYesNo)
      If gName = vbYes Then
         frmMain.Data1.Recordset.AddNew
         txtName.SetFocus
         frmMain.Data1.Recordset.Fields("Nombre") = txtName.Text
         frmMain.Data1.Recordset.Fields("Telefono") = txtPhoneNumber.Text
         frmMain.Data1.Recordset.Fields("[Llamado]") = Now()
         frmMain.Data1.Recordset.Update
         frmMain.Data1.Refresh
      Else
         'nada por ahora
      End If
   Else
      If (encontro2 = True) Then
      gName = MsgBox("El usuario ya existe..ingresarlo con otro telefono?", vbYesNo)
         If gName = vbYes Then
            frmMain.Data1.Recordset.AddNew
            txtName.SetFocus
            frmMain.Data1.Recordset.Fields("Nombre") = txtName.Text
            frmMain.Data1.Recordset.Fields("Telefono") = txtPhoneNumber.Text
            frmMain.Data1.Recordset.Fields("[Llamado]") = Now()
            frmMain.Data1.Recordset.Update
            frmMain.Data1.Refresh
         Else
            'nada por ahora
         End If
      Else
         frmMain.Data1.Recordset.AddNew
         txtName.SetFocus
```

```
        frmMain.Data1.Recordset.Fields("Nombre") = txtName.Text
        frmMain.Data1.Recordset.Fields("Telefono") = txtPhoneNumber.Text
        frmMain.Data1.Recordset.Fields("[Llamado]") = Now()
        frmMain.Data1.Recordset.Update
        frmMain.Data1.Refresh
      End If
   End If
mevoy:
Unload Me
End Sub
---------------------------------
Private Sub Form_Load()
   Me.Top = (Screen.Height / 2) - (Me.Height / 2)
   Me.Left = (Screen.Width / 2) - (Me.Width / 2)
End Sub
```

VB_Tapi.bas

```
Private udtLineDevCaps() As linedevcaps
Private cLineDevCapsExtra() As String * 2048
Type DialParams
   DeviceNumber As Integer
   DialableString As String
   Privilege As Long
   MediaMode As Long
End Type
Public Const LINEDIGITMODE_DTMF = &H2
Public Const TAPIVERSION = &H10004
Public Const LINECALLPRIVILEGE_NONE = &H1
Public Const LINECALLPRIVILEGE_MONITOR = &H2
Public Const LINECALLPRIVILEGE_OWNER = &H4
Public Const LINECALLINFOSTATE_CALLERID = 32768
Public Const LINECALLPARTYID_BLOCKED = &H1
Public Const LINECALLPARTYID_OUTOFAREA = &H2
Public Const LINECALLPARTYID_NAME = &H4
Public Const LINECALLPARTYID_ADDRESS = &H8
Public Const LINECALLPARTYID_PARTIAL = &H10
Public Const LINECALLPARTYID_UNKNOWN = &H20
Public Const LINECALLPARTYID_UNAVAIL = &H40
Public Const LINEMEDIAMODE_UNKNOWN = &H2
Public Const LINEMEDIAMODE_INTERACTIVEVOICE = &H4
Public Const LINEMEDIAMODE_AUTOMATEDVOICE = &H8
Public Const LINEMEDIAMODE_DATAMODEM = &H10
Public Const LINEMEDIAMODE_G3FAX = &H20
Public Const LINEMEDIAMODE_TDD = &H40
Public Const LINEMEDIAMODE_G4FAX = &H80
Public Const LINEMEDIAMODE_DIGITALDATA = &H100
Public Const LINEMEDIAMODE_TELETEX = &H200
Public Const LINEMEDIAMODE_VIDEOTEX = &H400
Public Const LINEMEDIAMODE_TELEX = &H800
Public Const LINEMEDIAMODE_MIXED = &H1000
Public Const LINEMEDIAMODE_ADSI = &H2000
Public Const LINEMEDIAMODE_VOICEVIEW = &H4000                    ' TAPI v1.4
Public Const LINECALLSELECT_LINE = &H1
Public Const LINECALLSELECT_ADDRESS = &H2
Public Const LINECALLSELECT_CALL = &H4
Public Enum TapiEvent
 LINE_ADDRESSSTATE = 0
 LINE_CALLINFO
 LINE_CALLSTATE
 LINE_CLOSE
 LINE_DEVSPECIFIC
 LINE_DEVSPECIFICFEATURE
 LINE_GATHERDIGITS
 LINE_GENERATE
 LINE_LINEDEVSTATE
 LINE_MONITORDIGITS
 LINE_MONITORMEDIA
 LINE_MONITORTONE
 LINE_REPLY
 LINE_REQUEST
 PHONE_BUTTON
 PHONE_CLOSE
 PHONE_DEVSPECIFIC
 PHONE_REPLY
 PHONE_STATE
 LINE_CREATE                          ' TAPI v1.4
 PHONE_CREATE                         ' TAPI v1.4
End Enum
Public Const LINECALLSTATE_IDLE = &H1
Public Const LINECALLSTATE_OFFERING = &H2
```

```
Public Const LINECALLSTATE_ACCEPTED = &H4
Public Const LINECALLSTATE_DIALTONE = &H8
Public Const LINECALLSTATE_DIALING = &H10
Public Const LINECALLSTATE_RINGBACK = &H20
Public Const LINECALLSTATE_BUSY = &H40
Public Const LINECALLSTATE_SPECIALINFO = &H80
Public Const LINECALLSTATE_CONNECTED = &H100
Public Const LINECALLSTATE_PROCEEDING = &H200
Public Const LINECALLSTATE_ONHOLD = &H400
Public Const LINECALLSTATE_CONFERENCED = &H800
Public Const LINECALLSTATE_ONHOLDPENDCONF = &H1000
Public Const LINECALLSTATE_ONHOLDPENDTRANSFER = &H2000
Public Const LINECALLSTATE_DISCONNECTED = &H4000
Public Const LINECALLSTATE_UNKNOWN = &H8000
Public Const LINEERR_ALLOCATED = &H80000001
Public Const LINEERR_BADDEVICEID = &H80000002
Public Const LINEERR_BEARERMODEUNAVAIL = &H80000003
Public Const LINEERR_CALLUNAVAIL = &H80000005
Public Const LINEERR_COMPLETIONOVERRUN = &H80000006
Public Const LINEERR_CONFERENCEFULL = &H80000007
Public Const LINEERR_DIALBILLING = &H80000008
Public Const LINEERR_DIALDIALTONE = &H80000009
Public Const LINEERR_DIALPROMPT = &H8000000A
Public Const LINEERR_DIALQUIET = &H8000000B
Public Const LINEERR_INCOMPATIBLEAPIVERSION = &H8000000C
Public Const LINEERR_INCOMPATIBLEEXTVERSION = &H8000000D
Public Const LINEERR_INIFILECORRUPT = &H8000000E
Public Const LINEERR_INUSE = &H8000000F
Public Const LINEERR_INVALADDRESS = &H80000010
Public Const LINEERR_INVALADDRESSID = &H80000011
Public Const LINEERR_INVALADDRESSMODE = &H80000012
Public Const LINEERR_INVALADDRESSSTATE = &H80000013
Public Const LINEERR_INVALAPPHANDLE = &H80000014
Public Const LINEERR_INVALAPPNAME = &H80000015
Public Const LINEERR_INVALBEARERMODE = &H80000016
Public Const LINEERR_INVALCALLCOMPLMODE = &H80000017
Public Const LINEERR_INVALCALLHANDLE = &H80000018
Public Const LINEERR_INVALCALLPARAMS = &H80000019
Public Const LINEERR_INVALCALLPRIVILEGE = &H8000001A
Public Const LINEERR_INVALCALLSELECT = &H8000001B
Public Const LINEERR_INVALCALLSTATE = &H8000001C
Public Const LINEERR_INVALCALLSTATELIST = &H8000001D
Public Const LINEERR_INVALCARD = &H8000001E
Public Const LINEERR_INVALCOMPLETIONID = &H8000001F
Public Const LINEERR_INVALCONFCALLHANDLE = &H80000020
Public Const LINEERR_INVALCONSULTCALLHANDLE = &H80000021
Public Const LINEERR_INVALCOUNTRYCODE = &H80000022
Public Const LINEERR_INVALDEVICECLASS = &H80000023
Public Const LINEERR_INVALDEVICEHANDLE = &H80000024
Public Const LINEERR_INVALDIALPARAMS = &H80000025
Public Const LINEERR_INVALDIGITLIST = &H80000026
Public Const LINEERR_INVALDIGITMODE = &H80000027
Public Const LINEERR_INVALDIGITS = &H80000028
Public Const LINEERR_INVALEXTVERSION = &H80000029
Public Const LINEERR_INVALGROUPID = &H8000002A
Public Const LINEERR_INVALLINEHANDLE = &H8000002B
Public Const LINEERR_INVALLINESTATE = &H8000002C
Public Const LINEERR_INVALLOCATION = &H8000002D
Public Const LINEERR_INVALMEDIALIST = &H8000002E
Public Const LINEERR_INVALMEDIAMODE = &H8000002F
Public Const LINEERR_INVALMESSAGEID = &H80000030
Public Const LINEERR_INVALPARAM = &H80000032
Public Const LINEERR_INVALPARKID = &H80000033
Public Const LINEERR_INVALPARKMODE = &H80000034
Public Const LINEERR_INVALPOINTER = &H80000035
Public Const LINEERR_INVALPRIVSELECT = &H80000036
```

```
Public Const LINEERR_INVALRATE = &H80000037
Public Const LINEERR_INVALREQUESTMODE = &H80000038
Public Const LINEERR_INVALTERMINALID = &H80000039
Public Const LINEERR_INVALTERMINALMODE = &H8000003A
Public Const LINEERR_INVALTIMEOUT = &H8000003B
Public Const LINEERR_INVALTONE = &H8000003C
Public Const LINEERR_INVALTONELIST = &H8000003D
Public Const LINEERR_INVALTONEMODE = &H8000003E
Public Const LINEERR_INVALTRANSFERMODE = &H8000003F
Public Const LINEERR_LINEMAPPERFAILED = &H80000040
Public Const LINEERR_NOCONFERENCE = &H80000041
Public Const LINEERR_NODEVICE = &H80000042
Public Const LINEERR_NODRIVER = &H80000043
Public Const LINEERR_NOMEM = &H80000044
Public Const LINEERR_NOREQUEST = &H80000045
Public Const LINEERR_NOTOWNER = &H80000046
Public Const LINEERR_NOTREGISTERED = &H80000047
Public Const LINEERR_OPERATIONFAILED = &H80000048
Public Const LINEERR_OPERATIONUNAVAIL = &H80000049
Public Const LINEERR_RATEUNAVAIL = &H8000004A
Public Const LINEERR_RESOURCEUNAVAIL = &H8000004B
Public Const LINEERR_REQUESTOVERRUN = &H8000004C
Public Const LINEERR_STRUCTURETOOSMALL = &H8000004D
Public Const LINEERR_TARGETNOTFOUND = &H8000004E
Public Const LINEERR_TARGETSELF = &H8000004F
Public Const LINEERR_UNINITIALIZED = &H80000050
Public Const LINEERR_USERUSERINFOTOOBIG = &H80000051
Public Const LINEERR_REINIT = &H80000052
Public Const LINEERR_ADDRESSBLOCKED = &H80000053
Public Const LINEERR_BILLINGREJECTED = &H80000054
Public Const LINEERR_INVALFEATURE = &H80000055
Public Const LINEERR_NOMULTIPLEINSTANCE = &H80000056
Public Const LINEERR_INVALAGENTID = &H80000057                ' TAPI v2.0
Public Const LINEERR_INVALAGENTGROUP = &H80000058             ' TAPI v2.0
Public Const LINEERR_INVALPASSWORD = &H80000059               ' TAPI v2.0
Public Const LINEERR_INVALAGENTSTATE = &H8000005A             ' TAPI v2.0
Public Const LINEERR_INVALAGENTACTIVITY = &H8000005B          ' TAPI v2.0
Public Const LINEERR_DIALVOICEDETECT = &H8000005C             ' TAPI v2.0
Public Const LINEDEVSTATE_OTHER = &H1
Public Const LINEDEVSTATE_RINGING = &H2
Public Const LINEDEVSTATE_CONNECTED = &H4
Public Const LINEDEVSTATE_DISCONNECTED = &H8
Public Const LINEDEVSTATE_MSGWAITON = &H10
Public Const LINEDEVSTATE_MSGWAITOFF = &H20
Public Const LINEDEVSTATE_INSERVICE = &H40
Public Const LINEDEVSTATE_OUTOFSERVICE = &H80
Public Const LINEDEVSTATE_MAINTENANCE = &H100
Public Const LINEDEVSTATE_OPEN = &H200
Public Const LINEDEVSTATE_CLOSE = &H400
Public Const LINEDEVSTATE_NUMCALLS = &H800
Public Const LINEDEVSTATE_NUMCOMPLETIONS = &H1000
Public Const LINEDEVSTATE_TERMINALS = &H2000
Public Const LINEDEVSTATE_ROAMMODE = &H4000
Public Const LINEDEVSTATE_BATTERY = &H8000
Public Const LINEDEVSTATE_SIGNAL = &H10000
Public Const LINEDEVSTATE_DEVSPECIFIC = &H20000
Public Const LINEDEVSTATE_REINIT = &H40000
Public Const LINEDEVSTATE_LOCK = &H80000
Public Const LINEDEVSTATE_CAPSCHANGE = &H100000       ' TAPI v1.4
Public Const LINEDEVSTATE_CONFIGCHANGE = &H200000     ' TAPI v1.4
Public Const LINEDEVSTATE_TRANSLATECHANGE = &H400000  ' TAPI v1.4
Public Const LINEDEVSTATE_COMPLCANCEL = &H800000      ' TAPI v1.4
Public Const LINEDEVSTATE_REMOVED = &H1000000         ' TAPI v1.4
Type linedialparams
 dwDialPause As Long
 dwDialSpeed As Long
```

```
  dwDigitDuration As Long
  dwWaitForDialtone As Long
End Type
Type lineCallInfo
   dwTotalSize As Long
   dwNeededSize As Long
   dwUsedSize As Long
   hLine As Long
   dwLineDeviceID As Long
   dwAddressID As Long
   dwBearerMode As Long
   dwRate As Long
   dwMediaMode As Long
   dwAppSpecific As Long
   dwCallID As Long
   dwRelatedCallID As Long
   dwCallParamFlags As Long
   dwCallStates As Long
   dwMonitorDigitModes As Long
   dwMonitorMediaModes As Long
   DialParams As linedialparams
   dwOrigin As Long
   dwReason As Long
   dwCompletionID As Long
   dwNumOwners As Long
   dwNumMonitors As Long
   dwCountryCode As Long
   dwTrunk As Long
   dwCallerIDFlags As Long
   dwCallerIDSize As Long
   dwCallerIDOffset As Long
   dwCallerIDNameSize As Long
   dwCallerIDNameOffset As Long
   dwCalledIDFlags As Long
   dwCalledIDSize As Long
   dwCalledIDOffset As Long
   dwCalledIDNameSize As Long
   dwCalledIDNameOffset As Long
   dwConnectedIDFlags As Long
   dwConnectedIDSize As Long
   dwConnectedIDOffset As Long
   dwConnectedIDNameSize As Long
   dwConnectedIDNameOffset As Long
   dwRedirectionIDFlags As Long
   dwRedirectionIDSize As Long
   dwRedirectionIDOffset As Long
   dwRedirectionIDNameSize As Long
   dwRedirectionIDNameOffset As Long
   dwRedirectingIDFlags As Long
   dwRedirectingIDSize As Long
   dwRedirectingIDOffset As Long
   dwRedirectingIDNameSize As Long
   dwRedirectingIDNameOffset As Long
   dwAppNameSize As Long
   dwAppNameOffset As Long
   dwDisplayableAddressSize As Long
   dwDisplayableAddressOffset As Long
   dwCalledPartySize As Long
   dwCalledPartyOffset As Long
   dwCommentSize As Long
   dwCommentOffset As Long
   dwDisplaySize As Long
   dwDisplayOffset As Long
   dwUserUserInfoSize As Long
   dwUserUserInfoOffset As Long
   dwHighLevelCompSize As Long
```

```
    dwHighLevelCompOffset As Long
    dwLowLevelCompSize As Long
    dwLowLevelCompOffset As Long
    dwChargingInfoSize As Long
    dwChargingInfoOffset As Long
    dwTerminalModesSize As Long
    dwTerminalModesOffset As Long
    dwDevSpecificSize As Long
    dwDevSpecificOffset As Long
    bBytes(2000) As Byte
End Type
Type LINECALLPARAMS
    dwTotalSize As Long
    dwBearerMode As Long
    dwMinRate As Long
    dwMaxRate As Long
    dwMediaMode As Long
    dwCallParamFlags As Long
    dwAddressMode As Long
    dwAddressID As Long
    DialParams As linedialparams
    dwOrigAddressSize As Long
    dwOrigAddressOffset As Long
    dwDisplayableAddressSize As Long
    dwDisplayableAddressOffset As Long
    dwCalledPartySize As Long
    dwCalledPartyOffset As Long
    dwCommentSize As Long
    dwCommentOffset As Long
    dwUserUserInfoSize As Long
    dwUserUserInfoOffset As Long
    dwHighLevelCompSize As Long
    dwHighLevelCompOffset As Long
    dwLowLevelCompSize As Long
    dwLowLevelCompOffset As Long
    dwDevSpecificSize As Long
    dwDevSpecificOffset As Long
    mem As String * 2048 ' sugerido por mca y SDK...no se bien porque pero lo deje por si acaso...
End Type
Public Const LINECALLPARAMS_FIXEDSIZE = 112
Type lineextensionid
    dwExtensionID0 As Long
    dwExtensionID1 As Long
    dwExtensionID2 As Long
    dwExtensionID3 As Long
End Type
Type linedevcaps
    dwTotalSize As Long
    dwNeededSize As Long
    dwUsedSize As Long
    dwProviderInfoSize As Long
    dwProviderInfoOffset As Long
    dwSwitchInfoSize As Long
    dwSwitchInfoOffset As Long
    dwPermanentLineID As Long
    dwLineNameSize As Long
    dwLineNameOffset As Long
    dwStringFormat As Long
    dwAddressModes As Long
    dwNumAddresses As Long
    dwBearerModes As Long
    dwMaxRate As Long
    dwMediaModes As Long
    dwGenerateToneModes As Long
    dwGenerateToneMaxNumFreq As Long
    dwGenerateDigitModes As Long
```

```
    dwMonitorToneMaxNumFreq As Long
    dwMonitorToneMaxNumEntries As Long
    dwMonitorDigitModes As Long
    dwGatherDigitsMinTimeout As Long
    dwGatherDigitsMaxTimeout As Long
    dwMedCtlDigitMaxListSize As Long
    dwMedCtlMediaMaxListSize As Long
    dwMedCtlToneMaxListSize As Long
    dwMedCtlCallStateMaxListSize As Long
    dwDevCapFlags As Long
    dwMaxNumActiveCalls As Long
    dwAnswerMode As Long
    dwRingModes As Long
    dwLineStates As Long
    dwUUIAcceptSize As Long
    dwUUIAnswerSize As Long
    dwUUIMakeCallSize As Long
    dwUUIDropSize As Long
    dwUUISendUserUserInfoSize As Long
    dwUUICallInfoSize As Long
    MinDialParams As linedialparams
    MaxDialParams As linedialparams
    DefaultDialParams As linedialparams
    dwNumTerminals As Long
    dwTerminalCapsSize As Long
    dwTerminalCapsOffset As Long
    dwTerminalTextEntrySize As Long
    dwTerminalTextSize As Long
    dwTerminalTextOffset As Long
    dwDevSpecificSize As Long
    dwDevSpecificOffset As Long
    dwLineFeatures As Long                    ' TAPI v1.4
    bBytes(2000) As Byte
End Type
Type varString
    dwTotalSize As Long
    dwNeededSize As Long
    dwUsedSize As Long
    dwStringFormat As Long
    dwStringSize As Long
    dwStringOffset As Long
    bBytes(2000) As Byte 'mi truquito del lineGetID
End Type
Public Declare Function lineMonitorDigits Lib "Tapi32" (ByVal hCall As Long, _
    ByVal dwDigitModes As Long) As Long
Public Declare Function lineGenerateDigits Lib "Tapi32" (ByVal hCall As Long, _
    ByVal dwDigitMode As Long, ByVal lpszDigits As String, ByVal dwDuration _
    As Long) As Long
Public Declare Function lineGetCallInfo Lib "Tapi32" (ByVal hCall As Long, _
    ByRef lpCallInf As lineCallInfo) As Long
Public Declare Function lineInitialize Lib "Tapi32" (ByRef hTAPI As Long, _
    ByVal hInst As Long, ByVal fnPtr As Long, ByRef szAppName As Long, _
    ByRef dwNumLines As Long) As Long
Public Declare Function lineNegotiateAPIVersion Lib "Tapi32" _
    (ByVal hTAPI As Long, ByVal dwDeviceID As Long, _
    ByVal dwAPILowVersion As Long, ByVal dwAPIHighVersion As Long, _
    ByRef lpdwAPIVersion As Long, ByRef lpExtensionID As lineextensionid) _
    As Long
Public Declare Function lineOpen Lib "Tapi32" (ByVal hLineApp As Long, _
    ByVal dwDeviceID As Long, ByRef lphLine As Long, ByVal dwAPIVersion As _
    Long, ByVal dwExtVersion As Long, ByRef dwCallbackInstance As Long, _
    ByVal dwPrivileges As Long, ByVal dwMediaModes As Long, _
    ByRef lpCallParams As Long) As Long
Public Declare Function lineGetDevCaps Lib "Tapi32" (ByVal hLineApp As Long, _
    ByVal dwDeviceID As Long, ByVal dwAPIVersion As Long, ByVal dwExtVersion _
    As Long, ByRef lpLineDevCaps As linedevcaps) As Long
```

```
Public Declare Function lineSetStatusMessages Lib "Tapi32" (ByVal hLine As _
   Long, ByVal dwLineStates As Long, ByVal dwAddressStates As Long) As Long
Public Declare Function lineMakeCall Lib "Tapi32" (ByVal hLine As Long, _
   ByRef lphCall As Long, ByVal lpszDestAddress As String, _
   ByVal dwCountryCode As Long, ByVal lpCallParams As Long) As Long
Public Declare Function lineDrop Lib "Tapi32" (ByVal hCall As Long, _
   ByVal lpsUserUserInfo As String, ByVal dwSize As Long) As Long
Public Declare Function lineShutdown Lib "Tapi32" (ByVal hLineApp As Long) _
   As Long
Public Declare Function lineAnswer Lib "Tapi32" (ByVal hCall As Long, _
   ByRef lpsUserUserInfo As String, ByVal dwSize As Long) As Long
Public Declare Function lineGetID Lib "Tapi32" (ByVal hLine As Long, _
   ByVal dwAddressID As Long, ByVal hCall As Long, ByVal dwSelect As Long, _
   ByRef lpDevice As varString, ByVal lpszDeviceClass As String) As Long
Public Declare Function lineDeallocateCall Lib "Tapi32" (ByVal hCall As Long) _
   As Long
Public Declare Function lineConfigDialog Lib "TAPI32.DLL" (ByVal dwDeviceID As Long, ByVal hwndOwner As
Integer, ByVal lpszDeviceClass As String) As Long
Public Declare Function lineClose Lib "TAPI32.DLL" (ByVal hLine As Long) As Long
```

TapiError.bas

```
Public Sub ProcessTAPIError(lrc As Long)
Select Case lrc
  Case LINEERR_ALLOCATED
    DebugString 0, " LINEERR_ALLOCATED"

  Case LINEERR_BADDEVICEID:
    DebugString 0, " LINEERR_BADDEVICEID"

  Case LINEERR_BEARERMODEUNAVAIL:
    DebugString 0, " LINEERR_BEARERMODEUNAVAIL"

  Case LINEERR_CALLUNAVAIL:
    DebugString 0, " LINEERR_CALLUNAVAIL"

  Case LINEERR_COMPLETIONOVERRUN:
    DebugString 0, " LINEERR_COMPLETIONOVERRUN"

  Case LINEERR_CONFERENCEFULL:
    DebugString 0, " LINEERR_CONFERENCEFULL"

  Case LINEERR_DIALBILLING:
    DebugString 0, " LINEERR_DIALBILLING"

  Case LINEERR_DIALDIALTONE:
    DebugString 0, " LINEERR_DIALDIALTONE"

  Case LINEERR_DIALPROMPT:
    DebugString 0, " LINEERR_DIALPROMPT"

  Case LINEERR_DIALQUIET:
    DebugString 0, " LINEERR_DIALQUIET"

  Case LINEERR_INCOMPATIBLEAPIVERSION:
    DebugString 0, " LINEERR_INCOMPATIBLEAPIVERSION"

  Case LINEERR_INCOMPATIBLEEXTVERSION:
    DebugString 0, " LINEERR_INCOMPATIBLEEXTVERSION"

  Case LINEERR_INIFILECORRUPT:
    DebugString 0, " LINEERR_INIFILECORRUPT"

  Case LINEERR_INUSE:
    DebugString 0, " LINEERR_INUSE"

  Case LINEERR_INVALADDRESS:
    DebugString 0, " LINEERR_INVALADDRESS"

  Case LINEERR_INVALADDRESSID:
    DebugString 0, " LINEERR_INVALADDRESSID"

  Case LINEERR_INVALADDRESSMODE:
    DebugString 0, " LINEERR_INVALADDRESSMODE"

  Case LINEERR_INVALADDRESSSTATE:
    DebugString 0, " LINEERR_INVALADDRESSSTATE"

  Case LINEERR_INVALAPPHANDLE:
    DebugString 0, " LINEERR_INVALAPPHANDLE"

  Case LINEERR_INVALAPPNAME:
    DebugString 0, " LINEERR_INVALAPPNAME"

  Case LINEERR_INVALBEARERMODE:
```

```
    DebugString 0, " LINEERR_INVALBEARERMODE"

Case LINEERR_INVALCALLCOMPLMODE:
    DebugString 0, " LINEERR_INVALCALLCOMPLMODE"

Case LINEERR_INVALCALLHANDLE:
    DebugString 0, " LINEERR_INVALCALLHANDLE"

Case LINEERR_INVALCALLPARAMS:
    DebugString 0, " LINEERR_INVALCALLPARAMS"

Case LINEERR_INVALCALLPRIVILEGE:
    DebugString 0, " LINEERR_INVALCALLPRIVILEGE"

Case LINEERR_INVALCALLSELECT:
    DebugString 0, " LINEERR_INVALCALLSELECT"

Case LINEERR_INVALCALLSTATE:
    DebugString 0, " LINEERR_INVALCALLSTATE"

Case LINEERR_INVALCALLSTATELIST:
    DebugString 0, " LINEERR_INVALCALLSTATELIST"

Case LINEERR_INVALCARD:
    DebugString 0, " LINEERR_INVALCARD"

Case LINEERR_INVALCOMPLETIONID:
    DebugString 0, " LINEERR_INVALCOMPLETIONID"

Case LINEERR_INVALCONFCALLHANDLE:
    DebugString 0, " LINEERR_INVALCONFCALLHANDLE"

Case LINEERR_INVALCONSULTCALLHANDLE:
    DebugString 0, " LINEERR_INVALCONSULTCALLHANDLE"

Case LINEERR_INVALCOUNTRYCODE:
    DebugString 0, " LINEERR_INVALCOUNTRYCODE"

Case LINEERR_INVALDEVICECLASS:
    DebugString 0, " LINEERR_INVALDEVICECLASS"

Case LINEERR_INVALDEVICEHANDLE:
    DebugString 0, " LINEERR_INVALDEVICEHANDLE"

Case LINEERR_INVALDIALPARAMS:
    DebugString 0, " LINEERR_INVALDIALPARAMS"

Case LINEERR_INVALDIGITLIST:
    DebugString 0, " LINEERR_INVALDIGITLIST"

Case LINEERR_INVALDIGITMODE:
    DebugString 0, " LINEERR_INVALDIGITMODE"

Case LINEERR_INVALDIGITS:
    DebugString 0, " LINEERR_INVALDIGITS"

Case LINEERR_INVALEXTVERSION:
    DebugString 0, " LINEERR_INVALEXTVERSION"

Case LINEERR_INVALGROUPID:
    DebugString 0, " LINEERR_INVALGROUPID"

Case LINEERR_INVALLINEHANDLE:
    DebugString 0, " LINEERR_INVALLINEHANDLE"

Case LINEERR_INVALLINESTATE:
```

```
    DebugString 0, " LINEERR_INVALLINESTATE"

Case LINEERR_INVALLOCATION:
    DebugString 0, " LINEERR_INVALLOCATION"

Case LINEERR_INVALMEDIALIST:
    DebugString 0, " LINEERR_INVALMEDIALIST"

Case LINEERR_INVALMEDIAMODE:
    DebugString 0, " LINEERR_INVALMEDIAMODE"

Case LINEERR_INVALMESSAGEID:
    DebugString 0, " LINEERR_INVALMESSAGEID"

Case LINEERR_INVALPARAM:
    DebugString 0, " LINEERR_INVALPARAM"

Case LINEERR_INVALPARKID:
    DebugString 0, " LINEERR_INVALPARKID"

Case LINEERR_INVALPARKMODE:
    DebugString 0, " LINEERR_INVALPARKMODE"

Case LINEERR_INVALPOINTER:
    DebugString 0, " LINEERR_INVALPOINTER"

Case LINEERR_INVALPRIVSELECT:
    DebugString 0, " LINEERR_INVALPRIVSELECT"

Case LINEERR_INVALRATE:
    DebugString 0, " LINEERR_INVALRATE"

Case LINEERR_INVALREQUESTMODE:
    DebugString 0, " LINEERR_INVALREQUESTMODE"

Case LINEERR_INVALTERMINALID:
    DebugString 0, " LINEERR_INVALTERMINALID"

Case LINEERR_INVALTERMINALMODE:
    DebugString 0, " LINEERR_INVALTERMINALMODE"

Case LINEERR_INVALTIMEOUT:
    DebugString 0, " LINEERR_INVALTIMEOUT"

Case LINEERR_INVALTONE:
    DebugString 0, " LINEERR_INVALTONE"

Case LINEERR_INVALTONELIST:
    DebugString 0, " LINEERR_INVALTONELIST"

Case LINEERR_INVALTONEMODE:
    DebugString 0, " LINEERR_INVALTONEMODE"

Case LINEERR_INVALTRANSFERMODE:
    DebugString 0, " LINEERR_INVALTRANSFERMODE"

Case LINEERR_LINEMAPPERFAILED:
    DebugString 0, " LINEERR_LINEMAPPERFAILED"

Case LINEERR_NOCONFERENCE:
    DebugString 0, " LINEERR_NOCONFERENCE"

Case LINEERR_NODEVICE:
    DebugString 0, " LINEERR_NODEVICE"

Case LINEERR_NODRIVER:
```

```
    DebugString 0, " LINEERR_NODRIVER"

  Case LINEERR_NOMEM:
    DebugString 0, " LINEERR_NOMEM"

  Case LINEERR_NOREQUEST:
    DebugString 0, " LINEERR_NOREQUEST"

  Case LINEERR_NOTOWNER:
    DebugString 0, " LINEERR_NOTOWNER"

  Case LINEERR_NOTREGISTERED:
    DebugString 0, " LINEERR_NOTREGISTERED"

  Case LINEERR_OPERATIONFAILED:
    DebugString 0, " LINEERR_OPERATIONFAILED"

  Case LINEERR_OPERATIONUNAVAIL:
    DebugString 0, " LINEERR_OPERATIONUNAVAIL"

  Case LINEERR_RATEUNAVAIL:
    DebugString 0, " LINEERR_RATEUNAVAIL"

  Case LINEERR_RESOURCEUNAVAIL:
    DebugString 0, " LINEERR_RESOURCEUNAVAIL"

  Case LINEERR_REQUESTOVERRUN:
    DebugString 0, " LINEERR_REQUESTOVERRUN"

  Case LINEERR_STRUCTURETOOSMALL:
    DebugString 0, " LINEERR_STRUCTURETOOSMALL"

  Case LINEERR_TARGETNOTFOUND:
    DebugString 0, " LINEERR_TARGETNOTFOUND"

  Case LINEERR_TARGETSELF:
    DebugString 0, " LINEERR_TARGETSELF"

  Case LINEERR_UNINITIALIZED:
    DebugString 0, " LINEERR_UNINITIALIZED"

  Case LINEERR_USERUSERINFOTOOBIG:
    DebugString 0, " LINEERR_USERUSERINFOTOOBIG"

  Case LINEERR_REINIT:
    DebugString 0, " LINEERR_REINIT"

  Case LINEERR_ADDRESSBLOCKED:
    DebugString 0, " LINEERR_ADDRESSBLOCKED"

  Case LINEERR_BILLINGREJECTED:
    DebugString 0, " LINEERR_BILLINGREJECTED"

  Case LINEERR_INVALFEATURE:
    DebugString 0, " LINEERR_INVALFEATURE"

  Case LINEERR_NOMULTIPLEINSTANCE:
    DebugString 0, " LINEERR_NOMULTIPLEINSTANCE"
'****** parte de tapi 2.0
    Case LINEERR_INVALAGENTID          ' &H80000057    // TAPI v2.0
      DebugString 0, "LINEERR_INVALAGENTID"
    Case LINEERR_INVALAGENTGROUP       ' &H80000058    // TAPI v2.0
      DebugString 0, "LINEERR_INVALAGENTGROUP"
    Case LINEERR_INVALPASSWORD         ' &H80000059    // TAPI v2.0
      DebugString 0, "LINEERR_INVALPASSWORD"
    Case LINEERR_INVALAGENTSTATE       ' &H8000005A    // TAPI v2.0
```

```
    DebugString 0, "LINEERR_INVALAGENTSTATE"
  Case LINEERR_INVALAGENTACTIVITY          ' &H8000005B    // TAPI v2.0
    DebugString 0, "LINEERR_INVALAGENTACTIVITY"
  Case LINEERR_DIALVOICEDETECT             ' &H8000005C    // TAPI v2.0
    DebugString 0, "LINEERR_DIALVOICEDETECT"
  Case Else   'If you see this is it probably a programming error.
    DebugString 0, " Unknown TAPI Error"
  End Select
End Sub
-----------------
Public Function TapiErrMsg(dwMsg As Long) As String
  Dim strTemp As String
  strTemp = "Unknown TAPI Error!"
  Select Case dwMsg
    Case LINEERR_ALLOCATED                 ' &H80000001
      strTemp = "LINEERR_ALLOCATED"
    Case LINEERR_BADDEVICEID               ' &H80000002
      strTemp = "LINEERR_BADDEVICEID"
    Case LINEERR_BEARERMODEUNAVAIL         ' &H80000003
      strTemp = "LINEERR_BEARERMODEUNAVAIL"
    Case LINEERR_CALLUNAVAIL               ' &H80000005
      strTemp = "LINEERR_CALLUNAVAIL"
    Case LINEERR_COMPLETIONOVERRUN         ' &H80000006
      strTemp = "LINEERR_COMPLETIONOVERRUN"
    Case LINEERR_CONFERENCEFULL            ' &H80000007
      strTemp = "LINEERR_CONFERENCEFULL"
    Case LINEERR_DIALBILLING               ' &H80000008
      strTemp = "LINEERR_DIALBILLING"
    Case LINEERR_DIALDIALTONE              ' &H80000009
      strTemp = "LINEERR_DIALDIALTONE"
    Case LINEERR_DIALPROMPT                ' &H8000000A
      strTemp = "LINEERR_DIALPROMPT"
    Case LINEERR_DIALQUIET                 ' &H8000000B
      strTemp = "LINEERR_DIALQUIET"
    Case LINEERR_INCOMPATIBLEAPIVERSION    ' &H8000000C
      strTemp = "LINEERR_INCOMPATIBLEAPIVERSION"
    Case LINEERR_INCOMPATIBLEEXTVERSION    ' &H8000000D
      strTemp = "LINEERR_INCOMPATIBLEEXTVERSION"
    Case LINEERR_INIFILECORRUPT            ' &H8000000E
      strTemp = "LINEERR_INIFILECORRUPT"
    Case LINEERR_INUSE                     ' &H8000000F
      strTemp = "LINEERR_INUSE"
    Case LINEERR_INVALADDRESS              ' &H80000010
      strTemp = "LINEERR_INVALADDRESS"
    Case LINEERR_INVALADDRESSID            ' &H80000011
      strTemp = "LINEERR_INVALADDRESSID"
    Case LINEERR_INVALADDRESSMODE          ' &H80000012
      strTemp = "LINEERR_INVALADDRESSMODE"
    Case LINEERR_INVALADDRESSSTATE         ' &H80000013
      strTemp = "LINEERR_INVALADDRESSSTATE"
    Case LINEERR_INVALAPPHANDLE            ' &H80000014
      strTemp = "LINEERR_INVALAPPHANDLE"
    Case LINEERR_INVALAPPNAME              ' &H80000015
      strTemp = "LINEERR_INVALAPPNAME"
    Case LINEERR_INVALBEARERMODE           ' &H80000016
      strTemp = "LINEERR_INVALBEARERMODE"
    Case LINEERR_INVALCALLCOMPLMODE        ' &H80000017
      strTemp = "LINEERR_INVALCALLCOMPLMODE"
    Case LINEERR_INVALCALLHANDLE           ' &H80000018
      strTemp = "LINEERR_INVALCALLHANDLE"
    Case LINEERR_INVALCALLPARAMS           ' &H80000019
      strTemp = "LINEERR_INVALCALLPARAMS"
    Case LINEERR_INVALCALLPRIVILEGE        ' &H8000001A
      strTemp = "LINEERR_INVALCALLPRIVILEGE"
    Case LINEERR_INVALCALLSELECT           ' &H8000001B
      strTemp = "LINEERR_INVALCALLSELECT"
```

```
Case LINEERR_INVALCALLSTATE             ' &H8000001C
   strTemp = "LINEERR_INVALCALLSTATE"
Case LINEERR_INVALCALLSTATELIST         ' &H8000001D
   strTemp = "LINEERR_INVALCALLSTATELIST"
Case LINEERR_INVALCARD                  ' &H8000001E
   strTemp = "LINEERR_INVALCARD"
Case LINEERR_INVALCOMPLETIONID          ' &H8000001F
   strTemp = "LINEERR_INVALCOMPLETIONID"
Case LINEERR_INVALCONFCALLHANDLE        ' &H80000020
   strTemp = "LINEERR_INVALCONFCALLHANDLE"
Case LINEERR_INVALCONSULTCALLHANDLE     ' &H80000021
   strTemp = "LINEERR_INVALCONSULTCALLHANDLE"
Case LINEERR_INVALCOUNTRYCODE           ' &H80000022
   strTemp = "LINEERR_INVALCOUNTRYCODE"
Case LINEERR_INVALDEVICECLASS           ' &H80000023
   strTemp = "LINEERR_INVALDEVICECLASS"
Case LINEERR_INVALDEVICEHANDLE          ' &H80000024
   strTemp = "LINEERR_INVALDEVICEHANDLE"
Case LINEERR_INVALDIALPARAMS            ' &H80000025
   strTemp = "LINEERR_INVALDIALPARAMS"
Case LINEERR_INVALDIGITLIST             ' &H80000026
   strTemp = "LINEERR_INVALDIGITLIST"
Case LINEERR_INVALDIGITMODE             ' &H80000027
   strTemp = "LINEERR_INVALDIGITMODE"
Case LINEERR_INVALDIGITS                ' &H80000028
   strTemp = "LINEERR_INVALDIGITS"
Case LINEERR_INVALEXTVERSION            ' &H80000029
   strTemp = "LINEERR_INVALEXTVERSION"
Case LINEERR_INVALGROUPID               ' &H8000002A
   strTemp = "LINEERR_INVALGROUPID"
Case LINEERR_INVALLINEHANDLE            ' &H8000002B
   strTemp = "LINEERR_INVALLINEHANDLE"
Case LINEERR_INVALLINESTATE             ' &H8000002C
   strTemp = "LINEERR_INVALLINESTATE"
Case LINEERR_INVALLOCATION              ' &H8000002D
   strTemp = "LINEERR_INVALLOCATION"
Case LINEERR_INVALMEDIALIST             ' &H8000002E
   strTemp = "LINEERR_INVALMEDIALIST"
Case LINEERR_INVALMEDIAMODE             ' &H8000002F
   strTemp = "LINEERR_INVALMEDIAMODE"
Case LINEERR_INVALMESSAGEID             ' &H80000030
   strTemp = "LINEERR_INVALMESSAGEID"
Case LINEERR_INVALPARAM                 ' &H80000032
   strTemp = "LINEERR_INVALPARAM"
Case LINEERR_INVALPARKID                ' &H80000033
   strTemp = "LINEERR_INVALPARKID"
Case LINEERR_INVALPARKMODE              ' &H80000034
   strTemp = "LINEERR_INVALPARKMODE"
Case LINEERR_INVALPOINTER               ' &H80000035
   strTemp = "LINEERR_INVALPOINTER"
Case LINEERR_INVALPRIVSELECT            ' &H80000036
   strTemp = "LINEERR_INVALPRIVSELECT"
Case LINEERR_INVALRATE                  ' &H80000037
   strTemp = "LINEERR_INVALRATE"
Case LINEERR_INVALREQUESTMODE           ' &H80000038
   strTemp = "LINEERR_INVALREQUESTMODE"
Case LINEERR_INVALTERMINALID            ' &H80000039
   strTemp = "LINEERR_INVALTERMINALID"
Case LINEERR_INVALTERMINALMODE          ' &H8000003A
   strTemp = "LINEERR_INVALTERMINALMODE"
Case LINEERR_INVALTIMEOUT               ' &H8000003B
   strTemp = "LINEERR_INVALTIMEOUT"
Case LINEERR_INVALTONE                  ' &H8000003C
   strTemp = "LINEERR_INVALTONE"
Case LINEERR_INVALTONELIST              ' &H8000003D
   strTemp = "LINEERR_INVALTONELIST"
```

```
    Case LINEERR_INVALTONEMODE              ' &H8000003E
      strTemp = "LINEERR_INVALTONEMODE"
    Case LINEERR_INVALTRANSFERMODE          ' &H8000003F
      strTemp = "LINEERR_INVALTRANSFERMODE"
    Case LINEERR_LINEMAPPERFAILED           ' &H80000040
      strTemp = "LINEERR_LINEMAPPERFAILED"
    Case LINEERR_NOCONFERENCE               ' &H80000041
      strTemp = "LINEERR_NOCONFERENCE"
    Case LINEERR_NODEVICE                   ' &H80000042
      strTemp = "LINEERR_NODEVICE"
    Case LINEERR_NODRIVER                   ' &H80000043
      strTemp = "LINEERR_NODRIVER"
    Case LINEERR_NOMEM                      ' &H80000044
      strTemp = "LINEERR_NOMEM"
    Case LINEERR_NOREQUEST                  ' &H80000045
      strTemp = "LINEERR_NOREQUEST"
    Case LINEERR_NOTOWNER                   ' &H80000046
      strTemp = "LINEERR_NOTOWNER"
    Case LINEERR_NOTREGISTERED              ' &H80000047
      strTemp = "LINEERR_NOTREGISTERED"
    Case LINEERR_OPERATIONFAILED            ' &H80000048
      strTemp = "LINEERR_OPERATIONFAILED"
    Case LINEERR_OPERATIONUNAVAIL           ' &H80000049
      strTemp = "LINEERR_OPERATIONUNAVAIL"
    Case LINEERR_RATEUNAVAIL                ' &H8000004A
      strTemp = "LINEERR_RATEUNAVAIL"
    Case LINEERR_RESOURCEUNAVAIL            ' &H8000004B
      strTemp = "LINEERR_RESOURCEUNAVAIL"
    Case LINEERR_REQUESTOVERRUN             ' &H8000004C
      strTemp = "LINEERR_REQUESTOVERRUN"
    Case LINEERR_STRUCTURETOOSMALL          ' &H8000004D
      strTemp = "LINEERR_STRUCTURETOOSMALL"
    Case LINEERR_TARGETNOTFOUND             ' &H8000004E
      strTemp = "LINEERR_TARGETNOTFOUND"
    Case LINEERR_TARGETSELF                 ' &H8000004F
      strTemp = "LINEERR_TARGETSELF"
    Case LINEERR_UNINITIALIZED              ' &H80000050
      strTemp = "LINEERR_UNINITIALIZED"
    Case LINEERR_USERUSERINFOTOOBIG         ' &H80000051
      strTemp = "LINEERR_USERUSERINFOTOOBIG"
    Case LINEERR_REINIT                     ' &H80000052
      strTemp = "LINEERR_REINIT"
    Case LINEERR_ADDRESSBLOCKED             ' &H80000053
      strTemp = "LINEERR_ADDRESSBLOCKED"
    Case LINEERR_BILLINGREJECTED            ' &H80000054
      strTemp = "LINEERR_BILLINGREJECTED"
    Case LINEERR_INVALFEATURE               ' &H80000055
      strTemp = "LINEERR_INVALFEATURE"
    Case LINEERR_NOMULTIPLEINSTANCE         ' &H80000056
      strTemp = "LINEERR_NOMULTIPLEINSTANCE"
    ' TAPI 2.0 only
    Case LINEERR_INVALAGENTID               ' &H80000057    // TAPI v2.0
      strTemp = "LINEERR_INVALAGENTID"
    Case LINEERR_INVALAGENTGROUP            ' &H80000058    // TAPI v2.0
      strTemp = "LINEERR_INVALAGENTGROUP"
    Case LINEERR_INVALPASSWORD              ' &H80000059    // TAPI v2.0
      strTemp = "LINEERR_INVALPASSWORD"
    Case LINEERR_INVALAGENTSTATE            ' &H8000005A    // TAPI v2.0
      strTemp = "LINEERR_INVALAGENTSTATE"
    Case LINEERR_INVALAGENTACTIVITY         ' &H8000005B    // TAPI v2.0
      strTemp = "LINEERR_INVALAGENTACTIVITY"
    Case LINEERR_DIALVOICEDETECT            ' &H8000005C    // TAPI v2.0
      strTemp = "LINEERR_DIALVOICEDETECT"
  End Select
  TapiErrMsg = strTemp & " [" & Hex(dwMsg) & "]"
End Function
```

Wave.bas

```
Public Const CALLBACK_FUNCTION = &H30000
Public Const WAVE_FORMAT_QUERY = &H1
Public Const WAVE_MAPPED = &H4
Public Const SND_FILENAME = &H20000
Public Const MMIO_READ = &H0
Public Const MMIO_FINDCHUNK = &H10
Public Const MMIO_FINDRIFF = &H20
Public Const MM_WOM_DONE = &H3BD
Public Const MM_WIM_OPEN = &H3BE
Public Const MM_WIM_CLOSE = &H3BF
Public Const MM_WIM_DATA = &H3C0
Public Const WAVE_FORMAT_PCM = &H1
Public Const GMEM_FIXED = &H0
Public Const GMEM_ZEROINIT = &H40
Public Const GPTR = GMEM_FIXED Or GMEM_ZEROINIT
Private Type FileHeader
   dwRiff As Long
   dwFileSize As Long
   dwWave As Long
   dwFormat As Long
   dwFormatLength As Long
   wFormatTag As Integer
   nChannels As Integer
   nSamplesPerSec As Long
   nAvgBytesPerSec As Long
   nBlockAlign As Integer
   wBitsPerSample As Integer
   dwData As Long
   dwDataLength As Long
End Type
Type WAVEHDR
  lpData As Long
  dwBufferLength As Long
  dwBytesRecorded As Long
  dwUser As Long
  dwFlags As Long
  dwLoops As Long
  lpNext As Long
  Reserved As Long
End Type
Type WAVEINCAPS
  wMid As Integer
  wPid As Integer
  vDriverVersion As Long
  szPname As String *32
  dwFormats As Long
  wChannels As Integer
End Type
Type WAVEFORMAT
  wFormatTag As Integer
  nChannels As Integer
  nSamplesPerSec As Long
  nAvgBytesPerSec As Long
  nBlockAlign As Integer
  wBitsPerSample As Integer
  cbSize As Integer
End Type
Type mmioinfo
  dwFlags As Long
  fccIOProc As Long
  pIOProc As Long
  wErrorRet As Long
```

```
  htask As Long
  cchBuffer As Long
  pchBuffer As String
  pchNext As String
  pchEndRead As String
  pchEndWrite As String
  lBufOffset As Long
  lDiskOffset As Long
  adwInfo(4) As Long
  dwReserved1 As Long
  dwReserved2 As Long
  hmmio As Long
End Type
Type MMCKINFO
  ckid As Long
  ckSize As Long
  fccType As Long
  dwDataOffset As Long
  dwFlags As Long
End Type
Declare Function waveInOpen Lib "winmm.dll" (lphWaveIn As Long, _
  ByVal uDeviceID As Long, lpFormat As WAVEFORMAT, ByVal dwCallback As Long, _
  ByVal dwInstance As Long, ByVal dwFlags As Long) As Long
Declare Function waveInPrepareHeader Lib "winmm.dll" (ByVal hWaveIn As Long, _
  lpWaveInHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function waveInAddBuffer Lib "winmm.dll" (ByVal hWaveIn As Long, _
  lpWaveInHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function waveInUnprepareHeader Lib "winmm.dll" (ByVal hWaveIn As Long, _
  lpWaveInHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function waveInStart Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveInReset Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveInClose Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveInStop Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveOutOpen Lib "winmm.dll" (lphWaveIn As Long, ByVal _
  uDeviceID As Long, lpFormat As WAVEFORMAT, ByVal dwCallback As Long, _
  ByVal dwInstance As Long, ByVal dwFlags As Long) As Long
Declare Function waveOutPrepareHeader Lib "winmm.dll" (ByVal hWaveIn As Long, _
  lpWaveInHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function waveOutReset Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveOutStart Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveOutStop Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveOutPause Lib "winmm.dll" (ByVal hWaveOut As Long) As Long
Declare Function waveOutRestart Lib "winmm.dll" (ByVal hWaveOut As Long) As Long
Declare Function waveOutUnprepareHeader Lib "winmm.dll" (ByVal hWaveIn As Long, _
  lpWaveInHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function waveOutClose Lib "winmm.dll" (ByVal hWaveIn As Long) As Long
Declare Function waveOutGetDevCaps Lib "winmm.dll" Alias "waveInGetDevCapsA" _
  (ByVal uDeviceID As Long, lpCaps As WAVEINCAPS, ByVal uSize As Long) As Long
Declare Function waveOutGetNumDevs Lib "winmm.dll" () As Long
Declare Function waveOutGetErrorText Lib "winmm.dll" Alias "waveInGetErrorTextA" _
  (ByVal err As Long, ByVal lpText As String, ByVal uSize As Long) As Long
Declare Function waveOutAddBuffer Lib "winmm.dll" (ByVal hWaveIn As Long, _
  lpWaveInHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function waveOutWrite Lib "winmm.dll" (ByVal hWaveOut As Long, _
  lpWaveOutHdr As WAVEHDR, ByVal uSize As Long) As Long
Declare Function mmioClose Lib "winmm.dll" (ByVal hmmio As Long, ByVal uFlags _
  As Long) As Long
Declare Function mmioDescend Lib "winmm.dll" (ByVal hmmio As Long, lpck As _
  MMCKINFO, lpckParent As MMCKINFO, ByVal uFlags As Long) As Long
Declare Function mmioDescendParent Lib "winmm.dll" Alias "mmioDescend" (ByVal _
  hmmio As Long, lpck As MMCKINFO, ByVal x As Long, ByVal uFlags As Long) As Long
Declare Function mmioOpen Lib "winmm.dll" Alias "mmioOpenA" (ByVal szFileName _
  As String, lpmmioinfo As mmioinfo, ByVal dwOpenFlags As Long) As Long
Declare Function mmioRead Lib "winmm.dll" (ByVal hmmio As Long, ByVal pch As _
  Long, ByVal cch As Long) As Long
Declare Function mmioReadFormat Lib "winmm.dll" Alias "mmioRead" (ByVal hmmio _
```

```
  As Long, ByRef pch As WAVEFORMAT, ByVal cch As Long) As Long
Declare Function mmioStringToFOURCC Lib "winmm.dll" Alias "mmioStringToFOURCCA" _
  (ByVal sz As String, ByVal uFlags As Long) As Long
Declare Function mmioAscend Lib "winmm.dll" (ByVal hmmio As Long, lpck As _
  MMCKINFO, ByVal uFlags As Long) As Long
Declare Function apisndPlaySound Lib "winmm" Alias _
      "sndPlaySoundA" (ByVal filename As String, ByVal snd_async _
      As Long) As Long
Declare Function GlobalAlloc Lib "kernel32" (ByVal wFlags As Long, ByVal _
                            dwBytes As Long) As Long
Declare Function GlobalLock Lib "kernel32" (ByVal hmem As Long) As Long
Declare Function GlobalFree Lib "kernel32" (ByVal hmem As Long) As Long
Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" ( _
      pvDest As Any, ByVal pvSource As Any, ByVal lBytes As Long)
Declare Sub CopyStructFromPtr Lib "kernel32" Alias "RtlMoveMemory" (struct As _
  Any, ByVal ptr As Long, ByVal cb As Long)
Public whdr As WAVEHDR      'nuestro wave en cabezera
Public format As WAVEFORMAT 'para playing y recording
Dim rc As Long              'Global error code
Dim msg As String * 200     'para error message lookup para el Wave API
Public hWaveOut As Long     'mi Handle parta mi wave out device
Dim bufferIn As Long        'wave in buffer pointer
Public hmem As Long         'Allocated memory
Dim outHdr As WAVEHDR       'Wave out header
Public h_wavein As Long     'mi Handle para mi wave in device
Public numSamples As Long   'numero de samples to play
Public fFileLoaded As Boolean   'flag para error de carga de wave
Public fPlaying As Boolean  'si se esta corriendop un wave file
Public m_BPlayRec As Boolean    'True para playing, False para recording
Public m_FileName As String 'Recorded file name
---------------------------
Public Sub SaveToFileAsStream(sName As String)
On Error Resume Next
   Dim fh As FileHeader, Status As Long
   Dim fFile, File_1Holder() As Byte
   Dim nErr As Long
   fFile = FreeFile
   If dlgGreeting.Visible = True Then  'grabando el greeting
      Kill App.Path + "\" + sName
      Open App.Path + "\" + sName For Binary Access Write As #fFile
   Else
      Open App.Path + "\Messages\" + sName For Binary Access Write As #fFile
   End If
   fh.dwRiff = &H46464952         'RIFF
   fh.dwWave = &H45564157         'WAVE
   fh.dwFormat = &H20746D66       'fmt_chnk
   fh.dwFormatLength = 16
   fh.wFormatTag = 7
   fh.nChannels = 1
   fh.nSamplesPerSec = 8000
   fh.nAvgBytesPerSec = 8000
   fh.wBitsPerSample = 8
   fh.nBlockAlign = 1
   fh.dwData = &H61746164          '               // data_chnk
   fh.dwDataLength = whdr.dwBytesRecorded
   fh.dwFileSize = whdr.dwBytesRecorded + Len(fh)
   ReDim File_1Holder(whdr.dwBytesRecorded + 1)
   CopyMemory File_1Holder(0), whdr.lpData, whdr.dwBytesRecorded + 1
   Put #fFile, , fh
   Put #fFile, , File_1Holder()
   nErr = GlobalFree(whdr.lpData)
   If nErr <> 0 Then DebugString 2, "SaveToFileAsStream leaked memory"
   Close #fFile
End Sub
---------------------------
Sub waveOutProc(ByVal hwi As Long, ByVal uMsg As Long, ByVal dwInstance _
```

```
        As Long, ByRef hdr As WAVEHDR, ByVal dwParam2 As Long)
  If (uMsg = MM_WOM_DONE) Then
    fPlaying = False
  End If
End Sub
------------------------------
Sub waveInProc(ByVal hwi As Long, ByVal uMsg As Long, ByVal dwInstance _
        As Long, ByRef hdr As WAVEHDR, ByVal dwParam2 As Long)
   Select Case uMsg
      Case MM_WIM_OPEN
      Case MM_WIM_CLOSE
      Case MM_WIM_DATA
         whdr = hdr
         frmMain.CleanUp = 1
   End Select
End Sub
------------------------------
Sub CloseWaveOut()
   rc = waveOutReset(hWaveOut)
   If rc <> 0 Then DebugString 0, CStr(rc) & ": " & "waveOutReset Error"
   rc = waveOutUnprepareHeader(hWaveOut, outHdr, Len(outHdr))
   If rc <> 0 Then DebugString 0, CStr(rc) & ": " & "waveOutUnprepareHeader Error"
   rc = waveOutClose(hWaveOut)
   If rc <> 0 Then DebugString 0, CStr(rc) & ": " & "waveOutClose Error"
End Sub
------------------------------
Sub CloseWaveIn()
   rc = waveInReset(h_wavein)
   If rc <> 0 Then DebugString 0, CStr(rc) & ": " & "waveInReset Error"
   rc = waveInUnprepareHeader(h_wavein, whdr, Len(whdr))
   If rc <> 0 Then DebugString 0, CStr(rc) & ": " & "waveInUnprepareHeader Error"
   rc = waveInClose(h_wavein)
   If rc <> 0 Then DebugString 0, CStr(rc) & ": " & "waveInClose Error"
End Sub
------------------------------
Sub LoadFile(inFile As String)
Dim mmckinfoParentIn As MMCKINFO
Dim mmckinfoSubchunkIn As MMCKINFO
  Dim hmmioIn As Long
  Dim mmioinf As mmioinfo
  fFileLoaded = False
  If (inFile = "") Then
     GlobalFree (hmem)
     Exit Sub
  End If
  hmmioIn = mmioOpen(inFile, mmioinf, MMIO_READ)
  If hmmioIn = 0 Then
     err.Raise mmioinf.wErrorRet, "LoadFile", _
     "Error opening input file: " & App.Path & inFile
     Exit Sub
  End If
  mmckinfoParentIn.fccType = mmioStringToFOURCC("WAVE", 0)
  rc = mmioDescendParent(hmmioIn, mmckinfoParentIn, 0, MMIO_FINDRIFF)
  If (rc <> 0) Then
     rc = mmioClose(hmmioIn, 0)
     err.Raise -1, "LoadFile", "Not a WAVE file"
     Exit Sub
  End If
  mmckinfoSubchunkIn.ckid = mmioStringToFOURCC("fmt", 0)
  rc = mmioDescend(hmmioIn, mmckinfoSubchunkIn, mmckinfoParentIn, MMIO_FINDCHUNK)
  If (rc <> 0) Then
     rc = mmioClose(hmmioIn, 0)
     err.Raise -1, "LoadFile", "Couldn't get format chunk"
     Exit Sub
  End If
  rc = mmioReadFormat(hmmioIn, format, Len(format))
```

```
  If (rc = -1) Then
    rc = mmioClose(hmmioIn, 0)
    err.Raise -1, "LoadFile", "Error reading format"
    Exit Sub
  End If
  rc = mmioAscend(hmmioIn, mmckinfoSubchunkIn, 0)
  rc = mmioDescend(hmmioIn, mmckinfoSubchunkIn, mmckinfoParentIn, MMIO_FINDCHUNK)
  If (rc <> 0) Then
    rc = mmioClose(hmmioIn, 0)
    err.Raise -1, "LoadFile", "Couldn't get data chunk"
    Exit Sub
  End If
  GlobalFree hmem
  hmem = GlobalAlloc(&H40, mmckinfoSubchunkIn.ckSize)
  bufferIn = GlobalLock(hmem)
  rc = mmioRead(hmmioIn, bufferIn, mmckinfoSubchunkIn.ckSize)
  numSamples = mmckinfoSubchunkIn.ckSize / format.nBlockAlign
  rc = mmioClose(hmmioIn, 0)
  If rc <> 0 Then DebugString 2, "Error: " & rc & " in Wave->LoadFile"
  fFileLoaded = True
End Sub
------------------------------
Sub Play(ByVal soundcard As Integer)
Dim lFlags As Long
   If soundcard = -1 Then
      lFlags = CALLBACK_FUNCTION
   Else
      lFlags = CALLBACK_FUNCTION Or WAVE_MAPPED
   End If
   format.cbSize = 0   'truco..si se daña el grabnado o play
   rc = waveOutOpen(hWaveOut, soundcard, format, AddressOf waveOutProc, 0, lFlags)
   If (rc <> 0) Then
     GlobalFree (hmem)
     waveOutGetErrorText rc, msg, Len(msg)
     err.Raise rc, "Play", msg & ""
     Exit Sub
   End If
   outHdr.lpData = bufferIn
   outHdr.dwBufferLength = numSamples * format.nBlockAlign
   outHdr.dwFlags = 0
   outHdr.dwLoops = 0
   rc = waveOutPrepareHeader(hWaveOut, outHdr, Len(outHdr))
   If (rc <> 0) Then
     waveOutGetErrorText rc, msg, Len(msg)
     err.Raise rc, "Play", msg & ""
     Exit Sub
   End If
   rc = waveOutWrite(hWaveOut, outHdr, Len(outHdr))
   If (rc <> 0) Then
     GlobalFree (hmem)
   Else
     fPlaying = True
     frmMain.Timer1.Enabled = True
   End If
End Sub
------------------------------
Public Sub RecStart(nMaxLength As Long, dwDevice As Long, sFileName As String)
Dim lFlags As Long
Dim bufsz As Long
Dim hData As Long
   If dwDevice = -1 Then
      lFlags = CALLBACK_FUNCTION
   Else
      lFlags = CALLBACK_FUNCTION Or WAVE_MAPPED
   End If
   frmMain.CleanUp = 0
```

```
    m_FileName = sFileName
    format.wFormatTag = 7
    format.nChannels = 1
    format.wBitsPerSample = 8
    format.nSamplesPerSec = 8000
    format.nBlockAlign = 1
    format.nAvgBytesPerSec = 8000
    format.cbSize = 0
    Dim myNull As Long
    rc = waveInOpen(h_wavein, dwDevice, format, AddressOf waveInProc, 0, lFlags)
    If rc <> 0 Then
       err.Raise rc, "RecStart", "Can Not Open the Device"
       Exit Sub
    End If
    AddWaveInBuffer bufsz, whdr
    rc = waveInStart(h_wavein)
    If rc <> 0 Then
       err.Raise rc, "RecStart", "Error in waveInStart"
       Exit Sub
    End If
    frmMain.Timer1.Enabled = True
    DebugString 5, "RecStart Completed"
End Sub
-----------------------------
Private Sub AddWaveInBuffer(bufsz As Long, whdr As WAVEHDR)
    Dim nErr As Long
    Dim hData As Long
    hData = GlobalAlloc(GPTR, bufsz)
    If hData = 0 Then
       err.Raise nErr, "AddWaveInBuffer", "Can Not Allocate Memory"
       Exit Sub
    End If
    whdr.lpData = GlobalLock(hData)
    If whdr.lpData = 0 Then
       err.Raise nErr, "AddWaveInBuffer", "Can Not Lock Memory"
       Exit Sub
    End If
    whdr.dwBufferLength = bufsz
    nErr = waveInPrepareHeader(h_wavein, whdr, Len(whdr))
    If nErr <> 0 Then
       err.Raise nErr, "AddWaveInBuffer", "Error in waveInPrepareHeader"
       Exit Sub
    End If
    nErr = waveInAddBuffer(h_wavein, whdr, Len(whdr))
    If nErr <> 0 Then
       err.Raise nErr, "AddWaveInBuffer", "Error in waveInAddBuffer"
       Exit Sub
    End If
    whdr.dwUser = hData
End Sub
-----------------------------
Sub PausePlay()
    waveOutPause (hWaveOut)
End Sub
-----------------------------
Sub ResumePlay()
    waveOutRestart (hWaveOut)
End Sub
-----------------------------
Sub StopPlay()
   waveOutReset (hWaveOut)
End Sub
```

Bibliografía

Windows Telephony programming, Chris Sells (Addison Wesley 1998)

The Mapi, Sapi & Tapi Developers Guide, Michael Amundsen (1997)

Aprenda VisuaL Basic Ya, Michael Halvorson (Microsoft Press)

Modern Telephony, Mahmoud Harb (Microsoft Press 1999)

Basic Carrier Telephony, David Talley (Microsoft Press 1999)

Computing System Architecture in Computer Telephony, Edwin Magulies (Microsoft Press 1999)

Tapí Application Development Questions, members.tripod.com/tapifaq6.html

Caller Id faq, www.ainslie.org.uk/callerid/cli-faq.htm

About Caller Id, www.markwelch.com/callerid.htm