

# 7. ACCESO A BASES DE DATOS LOCALES: BDE/IDAPI Y ODBC

## 7.1 IDAPI/BDE

### 7.1.1 Introducción

La mayoría de los sistemas que hacen uso de las Tecnologías del Habla para proporcionar servicios de valor añadido necesitan acceder a bases de datos. Un sistema de generación de aplicaciones telefónicas que no contemple este aspecto estaría muy limitado en su funcionalidad, puesto que no tendría acceso a más información de la que estuviera disponible en ficheros de texto o de voz almacenados en el propio PC sobre el que está implementada la aplicación telefónica.

El sistema monolínea contaba con la posibilidad de acceder a bases de datos a través del paquete software **BDE** (*Borland Database Engine*), diseñado para proporcionar a los diseñadores de aplicaciones Windows facilidades de acceso a múltiples bases de datos mediante una única API. La API que ofrece este paquete software se denomina **IDAPI** (*Integrated Database Application Program Interface*).

Las razones por las que, en su momento, se escogió IDAPI fueron las siguientes:

- Permite acceder a todo tipo de bases de datos, independientemente del formato en que estas estén creadas. Esto es fundamental, pues no parece lógico tener que implementar funciones ‘a medida’ para cada tipo de base de datos a la que queramos acceder.
- Permite acceder tanto a bases de datos locales como a otras residentes en un Host (remotas).
- Proporciona acceso directo a las fuentes de datos, sin necesidad de importarlos o exportarlos, o utilizar **DDE** (*Dynamic Data Exchange*).
- El acceso a las bases de datos se realiza a través de sentencias **SQL** (*Structured Query Language*) o **QBE** (*Query By Example*). Ambos lenguajes resultan muy útiles, cada uno por razones diferentes. SQL está muy extendido y no resulta excesivamente complicado de manejar. Respecto a QBE, resulta muy útil para usuarios sin conocimientos previos, permitiendo componer una consulta señalando sobre los campos de las tablas existentes en la base de datos, de forma visual. Gracias a estos lenguajes (utilizaremos

SQL en exclusiva) no sólo podremos realizar búsquedas sobre las bases de datos, sino también actualizaciones, inserciones, borrados, etc.

- Soporta **ODBC** (*Open DataBase Connectivity*).

Debido a todas estas ventajas se tomó la decisión de implementarlas en el sistema multilínea, aunque su funcionamiento queda restringido a una utilización monolínea.

Veamos algunos detalles necesarios para comprender su funcionamiento. Para más información consultar el proyecto de Mónica Fernández Pérez (ver *BIBLIOGRAFÍA*).

## 7.1.2 Arquitectura de BDE

BDE tiene una arquitectura basada en el manejo de diferentes *drivers*, uno por cada tipo de base de datos que se desee manejar. Además, se ha diseñado siguiendo una filosofía orientada a objetos, lo que facilita su ampliación. Si queremos que sea capaz de acceder a un tipo diferente de base de datos, basta instalar el *driver* BDE adecuado, o el *driver* ODBC correspondiente a la misma.

Además, este producto es muy útil en aplicaciones en entorno cliente/servidor, pues proporciona un acceso transparente a la aplicación, tanto a bases de datos locales como remotas.

## 7.1.3 IDAPI

IDAPI es la API de BDE. Está formada por un conjunto de funciones que pueden ser utilizadas por cualquier lenguaje de programación capaz de manejar DLLs de Windows, aunque está optimizado para trabajar con C/C++.

Al igual que BDE, IDAPI se ha diseñado según una filosofía orientada a objetos. Debido a esto, las aplicaciones que la manejen utilizan una serie de objetos, entre los que destacan:

- **Drivers:** cada *driver* es cargado automáticamente en el momento en que una aplicación requiere de éste una acción concreta. En ese momento, todos aquellos parámetros configurables relativos al mismo que se encuentren en el fichero de configuración de IDAPI se utilizan para inicializarlo.
- **Bases de datos:** una base de datos se maneja a través de un *handle* al objeto, que es creado en el momento en el que se le ordena a IDAPI abrir una base de datos.
- **Cursores:** son los que permiten acceder a los contenidos de las tablas de una base de datos o a los resultados de la ejecución de una consulta (*query*). Todas las operaciones de manipulación de datos y de posicionamiento en la

tabla se realizan a través del cursor. Se puede cerrar un cursor en cualquier momento, también es posible tener simultáneamente abiertos varios cursores sobre una misma tabla.

- **Queries:** estos objetos contienen un *query* (sentencia SQL válida) cuya validez ya ha sido comprobada por BDE, dispuesto a ser enviado para su ejecución.

## 7.1.4 Funciones de manejo de bases de datos basadas en IDAPI

Las funciones añadidas son todas funciones internas, y se encargan de gestionar el acceso a diferentes tipos de datos. Estas son las siguientes:

- **IDAPI\_ABRE\_BD:** abre la base de datos solicitada. Requiere los siguientes parámetros de entrada: nombre de la base de datos que se quiere abrir, tipo de base de datos de que se trata, y la clave de acceso a la base de datos, si es necesaria. Retorna 0 si la función se desarrolló correctamente y un valor negativo en caso de error.
- **IDAPI\_CIERRA\_BD:** cierra una base de datos. Requiere los siguientes parámetros de entrada: nombre de la base de datos a cerrar y su tipo. Retorna 0 si la función se desarrolló correctamente y un valor negativo en caso de error.
- **IDAPI\_BUSCAR:** ejecuta una sentencia SQL (consulta, actualización, inserción, borrado, etc.). Requiere los siguientes parámetros de entrada: nombre de la base de datos, tipo de la base de datos y el *query* a ser ejecutado. Retorna 0 si la función se desarrolló correctamente y un valor negativo en caso de error.
- **IDAPI\_LEER\_CAMPOS:** lee los datos almacenados en el cursor que devuelve IDAPI como resultado de una consulta a una base de datos. No requiere ningún parámetro de entrada y los parámetros de salida serán los datos correspondientes a la posición a la que está apuntando el cursor en ese momento. Retorna 0 si la función se desarrolló correctamente, 1 si era el último registro de datos disponible y un valor negativo en caso de error.

Estas funciones, en caso de producirse un error durante su ejecución, generan un fichero llamado 'DB\_error.txt' en el que se guarda información relativa a la función que falló, la causa del fallo y la fecha en que se produjo. Este fichero se crea, caso de no existir, en el directorio en que se esté ejecutando la aplicación telefónica.

En el caso de utilizarse el servidor IDAPI (se describe en el siguiente apartado), no son *funciones internas*, sino *funciones predefinidas*, pues es necesario que dispongan de función *idle* para realizar la espera no bloqueante de la respuesta del servidor. En este caso, la función *idle* se limita a esperar la respuesta del servidor, para proceder a leer los resultados, si lo hay, y el código de retorno de la función ejecutada por el servidor.

## 7.1.5 Desventajas de IDAPI. Servidor IDAPI

Aparte de la dependencia del paquete software *Borland Database Engine* (BDE), la realidad era que se estaba utilizando BDE para acceder vía ODBC (*Open DataBase Connectivity*); por lo que teníamos una capa software de la que no se utilizaba ningún servicio, ni se obtenía ningún beneficio.

Por otra parte, BDE/IDAPI no nos sirve para implementar un funcionamiento multilínea en el acceso a bases de datos, pues una llamada a cualquiera de sus funciones bloquea el sistema operativo hasta que se cursa la petición, por lo que nuestro sistema se queda en el punto en el que se produce la llamada, bloqueando al resto de las líneas.

Es de destacar que existe una función de *CALLBACK* en IDAPI que, mientras está procesando una petición, hace llamadas a una función definida por el usuario. Desgraciadamente, la dispersión de valores en los intervalos entre llamadas hace inviable su utilización para dar tiempo al sistema multilínea mientras se realiza el acceso a la base de datos.

Llegados a este punto se decidió ensayar una solución que había sido probada con éxito en el caso del acceso al Host IBM (se verá en el siguiente capítulo), consistente en crear un ejecutable independiente que actúe de servidor, con el objetivo de aprovechar la multitarea cooperativa de Windows. Todos los servidores desarrollados son multilínea, pero sólo atienden una línea a la vez. Es decir, si mientras están procesando una petición de una línea, llega otra petición de otra línea, no será atendida hasta que se acabe con la petición actual. Se podía haber realizado un servidor por línea, pero hay dos razones para no hacerlo:

- Sencillez y facilidad de expansión. Con el diseño actual no hay que hacer ninguna modificación para atender más líneas en el futuro.
- No existe una necesidad real, pues la única consecuencia negativa de la implementación adoptada es el pequeño retardo que se puede introducir, décimas de segundo, retardo que resulta imperceptible para el usuario.

La idea es que el sistema realice una petición al servidor y se quede esperando la respuesta, pero que esta espera no sea bloqueante. Es decir, que continúe atendiendo a todas las líneas y, de vez en cuando, consulte si ha llegado la respuesta.

De esta forma, el que se queda bloqueado es el servidor, pero si la llamada a la API de BDE/IDAPI es lo bastante cooperativa (da control a Windows de forma frecuente), y gracias a la multitarea cooperativa de Windows, el sistema sigue ejecutándose, y dado que no está realizando una espera bloqueante de la respuesta del servidor, será capaz de atender a todas las líneas que en ese momento estén activas.

La comunicación con el servidor se realiza a través de ficheros. Cuando el sistema (actúa como cliente) requiere un servicio del servidor, realiza una petición escribiendo un fichero de petición y otro de validación. La necesidad de escribir dos ficheros es evitar que el servidor detecte la presencia del fichero de petición y se disponga a abrirlo antes de que el cliente lo haya cerrado (el cliente siempre escribe primero el fichero de petición y después el de validación).

Cuando el servidor detecta la presencia de **ambos** ficheros, abre el fichero de petición, lee la petición concreta que se le hace, borra los dos ficheros y procede a realizar lo que se le pide. Cuando haya concluido escribirá los resultados, si los hay, y el código de retorno en un fichero de respuesta, y después escribirá el fichero de validación.

Mientras, el sistema ha estado iterando entre las líneas que esté atendiendo, en espera de la respuesta. Cuando detecta la presencia del fichero de respuesta, acompañado del fichero de validación, lee la respuesta y continúa normalmente.

Hay algunos detalles adicionales, básicamente por motivos de seguridad, para evitar situaciones anómalas que pudieran ocurrir:

- Cuando el sistema se dispone a hacer una petición (en la función *inic*), borra el fichero de respuesta, que no debería existir.
- El sistema no espera la respuesta (en la función *idle*) indefinidamente. Si esta no llega en un plazo razonable de tiempo, se devuelve un código de error. Es responsabilidad de la aplicación actuar en consecuencia, normalmente expulsando al usuario después de indicarle que se ha producido un fallo en el sistema.
- Cuando el servidor se dispone a escribir la respuesta de la petición que está cursando, comprueba si le ha llegado otra petición (de la misma línea). Si es así, no escribe la respuesta de la petición en curso y pasa a atender la petición que acaba de llegar. Este caso se puede dar si, mientras se está esperando una respuesta del servidor, el sistema detecta el *paso a falta*. Lo lógico es que el tratamiento de error asociado intente cerrar la base de datos que se está utilizando, para lo cual debe generar una petición. En este caso, el servidor se puede encontrar con esa petición cuando se dispone a escribir la respuesta a la petición anterior. Si el funcionamiento del sistema es el adecuado, éste es el único caso en el que se puede presentar esta situación, y la decisión de no escribir la respuesta de la petición en curso no tiene repercusiones, pues se va a cerrar la base de datos y el sistema, al detectar el *paso a falta*, habrá desactivado todas las funciones *idle* y volverá al inicio de la aplicación que esté ejecutando en espera de otra llamada.

Lamentablemente, una vez realizado el servidor, pudimos comprobar que la implementación de BDE/IDAPI es muy poco cooperativa, por lo que la solución del servidor no nos garantiza un funcionamiento multilínea en el acceso a bases de datos locales.

El paso a ODBC parecía una solución, pues en su documentación se indica que admite un modo asíncrono de funcionamiento, según el cual, tras realizar una petición, éste devuelve el control al sistema operativo. La aplicación que ha hecho la petición sólo debe consultar de vez en cuando al *driver* ODBC, en espera de que se complete su solicitud, para recoger entonces los resultados.

El problema es que los *drivers* ODBC de que disponemos no admiten el funcionamiento asíncrono, por lo que esta solución tampoco es viable. Se decidió intentar la solución del servidor (su arquitectura y el protocolo de comunicación entre cliente y

servidor vía ficheros son idénticos al caso de BDE/IDAPI). La conclusión es que ODBC no es lo bastante cooperativo, aunque si más que IDAPI, como para poder garantizar un funcionamiento adecuado del sistema multilínea.

Visto lo anterior, la única solución que parece viable es disponer de un sistema operativo con multitarea expulsiva, como Windows 95 o Windows NT. En el *Apéndice E* se describen algunos conceptos sobre Windows 95 que confirman este hecho y pueden ayudar a tomar la decisión en un futuro inmediato.

Se ha desarrollado una versión 32 bits del servidor ODBC (versión WIN32s, para la extensión de 32 bits de Windows 3.1, no válido para WIN32 que utiliza Windows 95). De esta forma, el futuro paso a entornos de 32 bits con multitarea expulsiva, como Windows 95 o Windows NT, será más fácil.

## 7.2 ODBC

### 7.2.1 Características de ODBC

ODBC es una interfaz de programación de aplicaciones estándar (API) que permite acceder a datos contenidos y manejados por sistemas de gestión de bases de datos (DBMSs). Utilizando ODBC, las aplicaciones pueden acceder a datos almacenados en una gran variedad de ordenadores personales, miniordenadores y grandes ordenadores, incluso aunque cada DBMS utilice un formato diferente para guardar la información.

Entre sus características, destacan:

- ODBC es una interfaz de programación de aplicaciones estándar que utiliza SQL (*Structured Query Language*).
- Oculta al programador la complejidad a la hora de conectarse a un origen de datos: por ejemplo, el acceso a los datos a través de redes de comunicación es transparente.
- Permite a múltiples aplicaciones acceder a múltiples orígenes de datos.
- Proporciona un modelo de programación *homogéneo*, es decir, bases de datos muy diferentes se manejan, vía ODBC, como si fueran idénticas, siendo ODBC el encargado de realizar las adaptaciones necesarias.
- Se basa en el modelo cliente/servidor.

### 7.2.2 Arquitectura de ODBC

Se basa en cuatro componentes:

- **Aplicaciones:** son las responsables de interactuar con el usuario y de llamar a las funciones ODBC para ejecutar sentencias SQL y recoger los resultados.
- **El *driver manager*:** se encarga de cargar y llamar a los *drivers* según lo demanden las aplicaciones.
- ***Drivers*:** procesan las llamadas a las funciones ODBC, ejecutan sentencias SQL y devuelven los resultados a las aplicaciones. Son también responsables de interactuar con cualquier capa software necesaria para acceder a las fuentes de datos, como puede ser el software de red.
- **Orígenes de datos:** consisten en conjuntos de datos, más todo lo que pueda ser necesario para llegar hasta ellos; sistemas operativos, gestores de bases de datos, redes de comunicación, etc.

### 7.2.3 *Handles* en ODBC

Un *handle* no es más que una variable de una aplicación, en la cual el sistema operativo es capaz de guardar información sobre la aplicación y sobre alguno de los objetos que maneja dicha aplicación.

ODBC usa tres tipos de *handles*:

- **De sistema (*environment*)**: es el *handle* de contexto global. Todo programa que utilice ODBC comienza solicitándolo y acaba liberándolo. Sólo puede haber uno por aplicación.
- **De conexión (*connection*)**: maneja toda la información relativa a una conexión. Identifica el *driver* que debe ser utilizado al realizar una conexión y en las llamadas posteriores a funciones ODBC. Puesto que se permiten varias conexiones, una aplicación puede solicitar varios.
- **De sentencia (*statement*)**: se utiliza para manejar todo el procesamiento relativo a una sentencia SQL, desde su ejecución hasta la recogida de datos.

En Windows, los *handles* se utilizan para acceder a estructuras de datos de las cuales sólo Windows conoce los detalles. Esto también se cumple en ODBC: las aplicaciones nunca *miran* el contenido de los *handles*, y tampoco manipulan el contenido a que hacen referencia. Este concepto, conocido como *ocultación de la información*, es uno de los principios básicos de la programación orientada a objeto.

Todas las funciones ODBC usan un *handle* como primer parámetro.

### 7.2.4 Controladores y orígenes de datos

El controlador (*driver*) es un dispositivo intermedio entre los datos y el programa de acceso a dichos datos. Los controladores se almacenan en ficheros con extensión DLL (librerías dinámicas de Windows) que generalmente se copian en el directorio SYSTEM de Windows. Tiene que haber un controlador para cada formato de bases de datos que se quiere utilizar.

Los orígenes de datos (*data source*) son los ficheros o directorios específicos donde se encuentran los datos. En el caso de las bases de datos locales (dBase, Paradox, Access, etc.) el origen de datos únicamente incluye la localización de los datos, no siendo necesario que esté disponible el gestor propietario de dicho formato. Sin embargo, en los servidores SQL no sólo es necesario indicar donde se encuentran los datos, sino que además es necesario que esté disponible el propio programa servidor de datos (SQL Server, Oracle, SQL Base, etc.).

El concepto de origen de datos es independiente del controlador y de los datos. Por ejemplo, se podrían crear dos orígenes de datos diferentes para la misma base de

datos, uno de ellos configurado para usar la base de datos en modo sólo lectura y el otro con autorizaciones para leer y escribir en la base de datos.

Cuando se ejecuta el icono ODBC del Panel de Control de Windows, aparece una ventana con la lista de los orígenes de datos, mostrando el nombre del origen de datos y, entre paréntesis, el controlador asociado con cada uno. Esta ventana permite, además de conocer los orígenes de datos instalados y el controlador asociado con cada uno, configurar, añadir y borrar los orígenes de datos.

### 7.2.5 Gestión de los orígenes de datos

Para instalar un origen de datos, debe pulsar el botón *Agregar*. Lo primero que nos pide es que identifiquemos el controlador con el que se asociará el origen de datos que se está creando. Por supuesto, en el caso de que se trate de un origen para un controlador nuevo, primero debe instalarse el controlador, tal como se explica posteriormente, y luego instalar el origen de datos para dicho controlador. Una vez seleccionado el controlador, aparece una ventana en la que es necesario definir ciertas características del origen de datos. Estas características varían en función de las opciones soportadas por el formato del origen de datos, pero siempre debe indicarse un nombre propio para identificar el origen (que no tiene porque coincidir con el nombre del directorio o fichero donde se encuentren los datos), y un directorio o nombre de fichero que corresponda al lugar donde se almacenan los datos.

En la lista de orígenes de datos hay otros botones además de *Agregar*. El botón *Configurar* permite definir los valores fundamentales del origen de datos. Al activar este botón aparece la misma ventana que se utilizó para crear el origen de datos. El botón *Eliminar* borra el origen de datos seleccionado (no borra físicamente el fichero de la base de datos al que está asociado dicho origen). Y el botón *Opciones* permite establecer algunas opciones generales para toda la gestión ODBC. Entre esas opciones está la posibilidad de registrar las llamadas de ODBC en un fichero específico con extensión LOG. Desactivar esta opción dará un mejor rendimiento de velocidad a la aplicación que esté haciendo llamadas ODBC; sin embargo, registrar todo lo que ocurre puede ser útil cuando se produzcan errores y así poderlos documentar.

Toda la gestión de orígenes de datos se realiza mediante el fichero ODBC.INI. De hecho, el administrador de ODBC lo único que hace es gestionar el contenido de dicho fichero. En el fichero ODBC.INI se encuentra la sección general [*ODBC Data Sources*], que almacena la lista de los orígenes de datos instalados. A continuación se establece una sección para cada origen que guarda los valores de cada una: directorio, controlador, etc.

### 7.2.6 Gestión de controladores

Debe haber un controlador para cada formato de bases de datos que queramos gestionar. Para acceder a las opciones relativas a los controladores hay que pulsar el botón *Controladores* en la ventana principal del Administrador ODBC (la ventana con la lista de los orígenes de datos instalados).

Al pulsar dicho botón, aparece una ventana con la lista de los controladores ODBC instalados. En esa ventana existen botones para borrar e instalar controladores. La instalación de un controlador exige tener un disquete con dicho controlador. En la sección general [*ODBC Drivers*] del fichero ODBCINST.INI aparecen los controladores instalados. Después existe una sección para cada controlador donde se detallan parámetros como su situación en el disco.

## 7.2.7 Funciones internas de manejo de bases de datos basadas en ODBC

Permiten el acceso a bases de datos locales utilizando **ODBC** (*Open DataBase Connectivity*), permitiendo realizar inserciones, modificaciones, actualizaciones, borrados, etc, utilizando sentencias **SQL** (*Structured Query Language*). En el *Apéndice D* puede encontrar una introducción a SQL.

Todas las funciones, excepto **ODBC\_LEER\_CAMPOS**, requieren al menos dos parámetros, los mismos en todos los casos, que son los siguientes:

- El **nombre de la base de datos**, que debe coincidir con uno de los orígenes de datos dados de alta en el Administrador de ODBC, que se encuentra en el Panel de Control de Windows.
- El **tipo de la base de datos** especifica el gestor de bases de datos que se ha de utilizar (access, dBase, etc) y se ha mantenido por compatibilidad con IDAPI, pero en realidad no se utiliza, pues viene determinado por el parámetro anterior.

Las funciones añadidas se encargan de gestionar el acceso a diferentes tipos de datos. Estas son:

- **ODBC\_ABRE\_BD**: abre la base de datos solicitada. Requiere los siguientes parámetros de entrada: nombre de la base de datos que se quiere abrir, tipo de base de datos de que se trata, y la clave de acceso a la base de datos, si es necesaria. Retorna 0 si la función se desarrolló correctamente y un valor negativo en caso de error. **La clave** debe coincidir con la clave de acceso de la base de datos que se esté utilizando. Si no se ha definido clave de acceso, puede escribirse cualquier cosa
- **ODBC\_CIERRA\_BD**: cierra una base de datos. Requiere los siguientes parámetros de entrada: nombre de la base de datos a cerrar y su tipo. Retorna 0 si la función se desarrolló correctamente y un valor negativo en caso de error.
- **ODBC\_BUSCAR**: ejecuta un *query* SQL (consulta, actualización, inserción, etc.) sobre la base de datos activa. Requiere los siguientes parámetros de entrada: nombre de la base de datos, tipo de la base de datos y el *query* a ser

ejecutado. Retorna 0 si la función se desarrolló correctamente y un valor negativo en caso de error. Para saber si el *query* ejecutado contiene o no resultados se debe ejecutar a continuación la siguiente función.

- **ODBC\_LEER\_CAMPOS**: lee un registro de los datos resultado de una consulta a una base de datos. Dichos datos sólo estarán disponibles hasta que se vuelva a realizar otra consulta. No requiere ningún parámetro de entrada y los parámetros de salida serán los campos del registro resultado del *query*. Retorna 0 si la función se desarrolló correctamente, 1 si era el último registro de datos disponible y un valor negativo en caso de error.

En el caso de utilizarse el servidor ODBC, no son *funciones internas*, sino *funciones predefinidas*, pues es necesario que dispongan de función *idle*. En este caso, la función *idle* se limita a esperar la respuesta del servidor, para proceder a leer los resultados, si los hay, y el código de retorno de la función ejecutada por el servidor.

## 7.2.8 Consideraciones adicionales

A la hora de escribir los nombre de los campos y los valores que se asignan a estos, se debe tener en cuenta lo siguiente:

- Los nombre de los campos no aparecerán entre comillas de ningún tipo. Por **ejemplo**: mi columna.
- Si el contenido de un campo es un carácter o una cadena de caracteres, aparecerá entre comillas simples. Por **ejemplo**: 'mi\_contenido'.
- Si el contenido de un campo es un número no aparecerá entre comillas. Por **ejemplo**: 18.

Lo anterior se ha comprobado para bases de datos de tipo Access y dBase, pero puede no ser extensible a otros tipos de bases de datos.

En el caso de BDE/IDAPI el tratamiento es distinto según el tipo de base de datos a utilizar. Para más información, consultar el proyecto de Mónica Fernández Pérez (ver *BIBLIOGRAFÍA*).

## 7.2.9 Ejemplos de aplicaciones que utilizan funciones ODBC

La siguiente aplicación utiliza todas las funciones ODBC, en su versión sin servidor. El objetivo es mostrar la sintaxis de las funciones ODBC, pues la aplicación no realiza ninguna tarea útil. Después veremos la misma aplicación, pero en su versión con servidor.

```
SECCION_ERRORES:
```

```
/* Añadir una SECCION_ERRORES estándar, como la mostrada en el apartado A.5.3.2. del
Apéndice A Manual del Usuario. */
```

```
SECCION_SUBRUTINAS:
```

```
SUBRUTINA fallo_sistema:
```

```
    reproducir("fallosys");
```

```
    funcion_interna ODBC_CIERRA_BD("die", "dbase");
```

```
    fin_funcion_interna;
```

```
    colgar();
```

```
    goto ESPERAR ;
```

```
    retornar;
```

```
FIN_SUBRUTINA
```

```
SECCION_APLICACION:
```

```
INICIO:
```

```
    n_interrumpir = 1;                /* 1=SI, 0=NO */
```

```
    n_timeout_sistema = 40;          /* en minutos */
```

```
    n_MAX_NUM_INTENTOS = 5;          /* antes de expulsarle */
```

```
    n_max_tiempo_reco_palabra = 3;   /* en segundos */
```

```
    n_MAX_RING = 1;                  /* tonos de llamada antes de descolgar */
```

```
    s_tipo_reco="ambos";              /* tipo de reconocimiento */
```

```
    n_usa_pitido = 1;                 /* usar pitido al reconocer cadenas */
```

```
    s_DIR_ESPECIF=".\\src\\mensajes\\notasdie\\";
```

```
ESPERAR:
```

```
    n_segunda_vez = 0;
```

```
    esperar_llamada();
```

```
INICIALIZACION:
```

```
    funcion_interna ODBC_ABRE_BD("die","dbase","0");
```

```
    switch
```

```
        case "-2" :
```

```
        case "-3" :
```

```
        case "-4" :
```

```
            gosub fallo_sistema;;
```

```
            break;
```

```
    fin_funcion_interna;
```

```

BUSQUEDA_INICIAL:
    n_dni = 1234567;
    SPRINTF(s_dni_con_ceros,"%09ld",n_dni);

    /* APE, NOMB, NOTA, CODASIG y ESTADO son campos de la tabla telefon */
    STRCAT("select APE,NOMB,NOTA,CODASIG,ESTADO from telefon where DNI=",
    s_dni_con_ceros, """; s_query);

    /* 'die' es un origen de datos que debe figurar en el Administrador de ODBC de
    Windows */
    /* 'telefon' es el nombre de la tabla de la base de datos a la que aplicamos el query */
    funcion_interna ODBC_BUSCAR("die", "dbase",s_query);
    switch
        case "0" :
        case "1" :
            break;
        case "-2" :
        case "-5" :
            gosub fallo_sistema;
            break;
    fin_funcion_interna;

    /* deben aparecer tantas variables como campos vayamos a leer */
    funcion_interna ODBC_LEER_CAMPOS(;s_ape,s_nomb,s_notas,s_codasig,s_estado)
    switch
        case "0":
        case "1":
            break;
        case "-2":
        case "-5":
            gosub fallo_sistema;
            break;
    fin_funcion_interna;

TERMINAR:
    funcion_interna ODBC_CIERRA_BD("die", "dbase");
    fin_funcion_interna;
    goto INICIALIZACION;;

QUITAR:
    colgar();
    goto INICIO: ;

FIN:
    
```

A continuación se muestra la misma aplicación, pero en la versión que hace uso del servidor. Obsérvese como ahora las funciones no son internas, sino predefinidas, y para conocer su código de retorno se utiliza la *función interna RESULTADO\_ANTERIOR*.

SECCION\_ERRORES:

/\* Añadir una SECCION\_ERRORES estándar, como la mostrada en el apartado A.5.3.2. del Apéndice A Manual del Usuario. \*/

SECCION\_SUBRUTINAS:

SUBRUTINA fallo\_sistema:

```
reproducir("fallosys");
ODBC_CIERRA_BD("die", "dbase");
colgar();
goto ESPERAR; ;
retornar;
```

FIN\_SUBRUTINA

SECCION\_APLICACION:

INICIO:

```
n_interrumpir = 1; /* 1=SI, 0=NO */
n_timeout_sistema = -1; /* en minutos */
n_MAX_NUM_INTENTOS = 5; /* antes de expulsarle */
n_max_tiempo_reco_palabra = 3; /* en segundos */
n_MAX_RING = 1; /* tonos de llamada antes de descolgar */
s_tipo_reco="ambos"; /* tipo de reconocimiento */
n_usa_pitido = 1; /* usar pitido al reconocer cadenas */
s_DIR_ESPECIF=".\\src\\mensajes\\notasdie\\";
```

ESPERAR:

```
n_segunda_vez = 0;
esperar_llamada();
```

INICIALIZACION:

```
ODBC_ABRE_BD("die","dbase","0");
funcion_interna RESULTADO_ANTERIOR();
switch
    case "-2" :
    case "-3" :
    case "-4" :
        gosub fallo_sistema;;
        break;
fin_funcion_interna;
```

BUSQUEDA\_INICIAL:

```
n_dni = 1234567;
SPRINTF(s_dni_con_ceros,"%09ld",n_dni);
```

/\* APE, NOMB, NOTA, CODASIG y ESTADO son campos de la tabla telefon \*/

```
STRCAT("select APE,NOMB,NOTA,CODASIG,ESTADO from telefon where DNI = '",
s_dni_con_ceros,"''"; s_query);
```

/\* 'die' es un origen de datos que debe figurar en el Administrador de ODBC de Windows \*/  
/\* 'telefon' es el nombre de la tabla de la base de datos a la que aplicamos el query \*/

```
ODBC_BUSCAR("die", "dbase",s_query);
```

```

funcion_interna RESULTADO_ANTERIOR(;)
switch
    case "0" :
    case "1" :
        break;
    case "-2" :
    case "-5" :
        gosub fallo_sistema;;
        break;
fin_funcion_interna;

/* deben aparecer tantas variables como campos vayamos a leer */
ODBC_LEER_CAMPOS(;s_ape,s_nomb,s_notas,s_codasig,s_estado);
funcion_interna RESULTADO_ANTERIOR(;)
switch
    case "0" :
    case "1" :
        break;
    case "-2" :
    case "-5" :
        gosub fallo_sistema;;
        break;
fin_funcion_interna;

TERMINAR:
    ODBC_CIERRA_BD("die", "dbase");
    goto INICIALIZACION;;

QUITAR:
    colgar();
    goto INICIO;

FIN:

```

<b>7.</b>	<b>ACCESO A BASES DE DATOS LOCALES: BDE/IDAPI Y ODBC .....</b>	<b>7-1</b>
<b>7.1</b>	<b>IDAPI/BDE.....</b>	<b>7-1</b>
7.1.1	Introducción.....	7-1
7.1.2	Arquitectura de BDE.....	7-2
7.1.3	IDAPI.....	7-2
7.1.4	Funciones de manejo de bases de datos basadas en IDAPI .....	7-3
7.1.5	Desventajas de IDAPI. Servidor IDAPI.....	7-4
<b>7.2</b>	<b>ODBC .....</b>	<b>7-7</b>
7.2.1	Características de ODBC.....	7-7
7.2.2	Arquitectura de ODBC.....	7-7
7.2.3	<i>Handles</i> en ODBC .....	7-8
7.2.4	Controladores y orígenes de datos .....	7-8
7.2.5	Gestión de los orígenes de datos .....	7-9
7.2.6	Gestión de controladores .....	7-9
7.2.7	Funciones internas de manejo de bases de datos basadas en ODBC.....	7-10
7.2.8	Consideraciones adicionales.....	7-11
7.2.9	Ejemplos de aplicaciones que utilizan funciones ODBC.....	7-12