UNIVERSIDAD LASALLISTA

BENAVENTE

ESCUELA DE INGENIERIA EN COMPUTACION

Con Estudios Incorporados a la Universidad Nacional Autónoma de México

CLAVE: 8793-16

"Manual Teórico-Práctico Basado en Delphi 7"

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

PRESENTA:

JULIO NAVA ARREDONDO

ASESOR: ING. MAYA GICELA VILLAGOMEZ TORRES

CELAYA, GUANAJUATO. **AGOSTO DEL 2005**

AGRADECIMIENTOS

A Dios:

Por darme la oportunidad de terminar un reto más, por vivir, tener una gran familia y por darme la fortaleza para seguir adelante.

A mis Padres Julio y María:

Lo más valioso que tengo porque sin ellos no hubiera podido realizar mi carrera, por el apoyo que siempre tengo incondicional y enseñarme lo valioso de la vida, por el amor que les tengo esté gran triunfo es para ustedes. Gracias Papas

A mis Hermanas Angélica, Gabriela y Sobrina Lupita:

Quiero que sepan que las quiero mucho, gracias por su apoyo, sus consejos ya que con ellos me ayudaron a salir adelante y lograr mis metas.

A mi novia Lourdes:

Por su cariño, amor y apoyo en todo momento, eres parte de este momento tan especial para mí.

A mis amigos:

Por compartir grandes momentos juntos, por estar ahí en las buenas y malas y quiero que sepan que siempre tendrán una amigo en mí.

A mis maestros y asesor:

Por su apoyo en la realización de mi proyecto, por sus consejos y sabiduría que compartieron a lo largo de nuestra convivencia.

INDICE

IV. Programación Orientada a Objetos

V. La Biblioteca de Componentes Visuales (VCL/CLX)

VI. Administración de Proyectos

VII. Desarrollo de Aplicaciones Win32

VIII. Tecnologías de Acceso a Datos

IX. Desarrollo de Aplicaciones con dbExpress

X. Manejo de Excepciones y Depuración

XI. Creación de Reportes con Rave Reports

CONCLUSION

BIBLIOGRAFIA

INTRODUCCION

La programación a través de los años ha tenido un gran desempeño y de gran ayuda para la facilitación de los trabajos dentro de una empresa o institución, por eso presento este manual que comprende el lenguaje de programación Delphi, el cual se nos facilitará la comprensión, análisis y elaboración del problema.

Hoy en día en muy importante tener conocimiento de varios lenguajes de programación ya que con eso tenemos la facilidad de tener opción para elegir con cuál lenguaje de programación se nos facilitará el trabajo y también tener mas opción en cuestión de trabajo.

El alumno, maestro o cualquier persona puede aprender Delphi porque es un lenguaje basado en el lenguaje pascal, en este manual empezaremos desde lo básico que es muy importante para la compresión del lenguaje y utilerías con que cuenta.

En el capítulo I analizó lo que es Delphi, sus ediciones y sus características principales con las que cuenta, también analizó la historia como fue desarrollándose el Delphi a través de los años.

En el capítulo II muestro todas las ventanas, su significado y para que nos sirve cada una, el cual es de gran ayuda para el rápido manejo del lenguaje y sus utilerías.

En el capítulo III analizo en sí lo propio del lenguaje como su sintaxis, tipos de variables, tipos de datos etc, ya que éstas son las bases de cualquier lenguaje de programación y por consecuencia nos ayudará para iniciar la elaboración de nuestro programa y también nos ayudará para la materia de programación de sistemas.

En el capítulo IV analizo el significado de lo que es un lenguaje Orientado a Objetos así como también cada una que lo comprende como son Clases, Objetos, etc, y sus componentes que comprenden cada una.

En el capítulo V comprendo lo que llamamos componentes visuales y sus características, estos son los que comprenden cada una de las herramientas con que contamos para realizar el programa en forma visual como son los botones, frames, cuadros de texto, etc.

La administración de nuestro proyecto es un objetivo muy importante que nos ayuda para tener mejor organizado todos nuestros archivos de trabajo y facilitarnos el trabajo esto se analizará en el capítulo VI.

En el capítulo VII desarrollamos aplicaciones basadas en Win32 que nos serán de ayuda para comprender el funcionamiento de las mismas. Elaboraremos un ejercicio aplicando estos conceptos básicos de un editor de texto, el cual nos ayuda para la comprensión de lo que Win32.

Dentro de un sistema las bases de datos son de gran importancia que son analizo en el capítulo VIII, los tipos de bases de datos y sus características de cada una, por ejemplo Ttable, Tquery, Tdatabase, etc. este capítulo nos ayudará a tener material para la materia "bases de datos" de nuestro plan de estudios de Ingeniería, también es muy importante la manipulación de los datos dentro de los sistemas, analizaremos sus componentes visuales dentro del programa Delphi para mayor facilidad de ubicación. Elaboraremos ejercicios utilizando bases de datos y lo anteriormente visto.

Los errores dentro de nuestro proyecto nos ayudan a darnos cuenta de lo que está fallando por eso en el capítulo X explicamos cómo comprenderlos y depurarlos, también mostraremos algunas ventanas de ayuda que nos muestran el comportamiento de cada una las partes de nuestro programa como son las variables, watch list, etc.

Por último, para el usuario final es importante tener la información en papel por eso en el capítulo XI analizo cómo crear un reporte desde lo más sencillo hasta tener la información de varias bases de datos. Con esto tenemos comprendido gran parte de lo que es el lenguaje de programación Delphi esperando que sea de gran ayuda para toda persona que deseé aprenderlo.

Hoy en día es muy importante que los alumnos salgan mejor preparados por eso es una gran opción impartir el lenguaje dentro de los talleres, también en la materia de ingeniería en programación y seminario de ingeniería en computación que son impartidos a los estudiantes de Ingeniería en computación de la Universidad Lasallista Benavente.

JUSTIFICACION

Hoy en día uno de los puntos más importantes es la preparación por eso decidí la elaboración de un manual que consta de un lenguaje de programación que es **Delphi** porque es la herramienta de mi trabajo y se me hizo interesante compartir mis conocimientos con todos los alumnos para que tengan una herramienta más conque enfrentarse en la vida cotidiana.

Además influyó para que escogiera este tema es que la base del **Delphi** está constituida con un lenguaje estructurado llamado Pascal, lo cual me facilitará la comprensión del mismo.

También lo justifico porque me ayudó mucho a comprender con mayor facilidad todo lo que puedo desarrollar dentro de mi trabajo y todo las herramientas con las que cuenta este lenguaje de programación.

‘’DESARROLLO DEL MANUAL TEORICO-PRACTICO BASADO EN DELPHI 7’’

OBJETIVO GENERAL

Crear y desarrollar aplicaciones basado en el manual teórico-práctico de delphi 7 y comprender su gran variedad de utilerías con las que cuenta, que será de gran ayuda para el alumno que desee aprenderlo.

OBJETIVO ESPECIFICO

- Manipular el Ambiente Integrado de Desarrollo de Delphi para el desarrollo rápido de aplicaciones para Windows.
- Conocer la sintaxis y riqueza del lenguaje Objeto Pascal para la construcción modular de programas, unidades y librerías.
- Comprender los conceptos básicos de la Programación Orientada a Objetos.
- Conocer y utilizar los componentes de uso frecuente de la Biblioteca de Componentes Visuales que le permitan diseñar de manera rápida, atractiva e intuitiva interfaces gráficas de usuario.
- Conocer el depurador de aplicaciones para la construcción de aplicaciones robustas.
- Entender la arquitectura de acceso a datos para construir aplicaciones que consulten y manipulen información de bases de datos.
- Utilizar Rave Reports para generar reportes desde sus aplicaciones Delphi.

HIPOTESIS

El alumno contará con los conocimientos necesarios para la elaboración de cualquier sistema, para la automatización de cualquier proceso por medio del Software **Delphi,** ya que es muy versátil y amigable para el usuario, y puede ser muy fácil para programar.

I. Conocimiento Delphi

1.1 ¿Qué es Delphi?

"Delphi es básicamente un compilador de pascal cuya historia data del primer Turbo Pascal. Cuenta con un Ambiente Integrado de Desarrollo, permite un Desarrollo Rápido de Aplicaciones (RAD), proporciona capacidades de manejo de bases de datos, entre otras características que lo hacen único. En esencia Delphi 7 es la culminación de casi dos décadas de desarrollo por parte de Borland, proporcionando estabilidad en un compilador de 32 bits optimizado".[1]

1.2 Historia sobre la evolución de Delphi

Delphi comenzó como el nombre código de un proyecto beta que se estaba desarrollando en Borland, en cual consistía en un entorno de desarrollo visual de próxima generación para Windows basado en el lenguaje de programación Object Pascal. El nombre surgió a mediados de 1993, después de seis meses de investigación y análisis de mercado.

Se había tomado la decisión de que el nuevo producto tuviese herramientas y conectividad de bases de datos como parte central del mismo, así que se buscaba un nombre código que reflejara este hecho; el desarrollar Delphi como un producto con soporte de bases de datos era precisamente lo que se necesitaba para romper la barrera entre las herramientas de Pascal de Borland y Visual Basic, el objetivo final era que Delphi estuviera por encima de las herramientas tradicionales de desarrollo en Windows.

Uno de los primeros nombre código sugeridos para el nuevo producto era "Oracle", por razones obvias éste se descartó, pero surgió la idea de que en la antigua Grecia para obtener información de debía consultar al Oráculo, el cual se encontraba en la cuidad de Delphi; el nombre tomó arraigo pues al ser Pascal un lenguaje de programación clásico, era practico asociar una herramienta de desarrollo basada en Pascal con una imagen clásica de la mitología griega.

Cuando llegó el momento de seleccionar un nombre para el producto final, se sugirieron nombres que representaran la "funcionalidad del producto", entre estos nombres se encontraban AppBuilder. Este nombre aún aparece en algunas de las clases internas del IDE, en concreto, en el nombre de la clase de la ventana principal del IDE, pues el equipo de desarrollo tenia presiones para implementar esta clase, dicho nombre no prosperó, en parte porque principalmente se descartó porque Novell liberó en ese entonces un producto llamada Visual AppBuilder. Por estas razones se utilizó "Delphi" como el nombre final del producto.

[1] Ayuda del Software Delphi 7

Borland tiene un largo historial de nombres códigos inusuales, el nombre de Delphi 7 antes de su liberación fue "Aurora".

A través de los años Delphi ha evolucionado, a continuación se muestra un resumen de dicha evolución.

Delphi 1

En la época de DOS, los desarrolladores tenían a su disposición herramientas productivas pero lentas y el lenguaje ensamblador, que aunque eficiente era complejo.

En el año de 1993 Turbo Pascal fue la respuesta a la necesidad de contar con una herramienta productiva con el rendimiento de un compilador real, Borland licenció el núcleo del compilador de Pascal, escrito por Anders Hejlsberg, quien trabajaba en una compañía en Dinamarca y agregó la interfaz de usuario y el editor. Hejlsberg se unió a Borland y se convirtió en el arquitecto de todas las versiones de Turbo Pascal y de las primeras tres versiones de Delphi.

Años después con el advenimiento de Windows 3.11 los desarrolladores se enfrentaron a un problema similar, por un lado existían lenguajes poderosos pero de difícil manejo como C++ y por otro lado existían lenguajes fáciles de manejar pero limitados como Visual Basic.

El 14 de Febrero de 1995 Delphi 1 debutó en una presentación ante 1500 desarrolladores brindando un enfoque radicalmente diferente al desarrollo de aplicaciones en Windows. Delphi fue la primera herramienta de desarrollo en Windows que combinó un ambiente de desarrollo visual, con un compilador optimizado de código nativo y un motor escalable de acceso a datos, gracias a Delphi se acuñó la frase Desarrollo Rápido de Aplicaciones (RAD).

Delphi 2

Un año mas tarde Delphi 2 aportó mejoras como un compilador optimizado de 32 bits, soporte para OCX, componentes de interfaz de usuario de Windows 95, soporte para programación multi-hilos, un motor de base de datos de 32 bits, nuevos tipos de datos, un acoplamiento más cercano a C++, herencia visual de formas y un IDE mejorado.

Delphi 3

En 1997 Delphi 3 brindó soporte para ActiveX, permitiendo crear servidores COM e interfaces COM en una forma simple y confiable; permitió crear componentes de VCL a partir de controles ActiveX, además de la construcción de clientes delgados para el Word Wide Web y de aplicaciones de bases de datos de múltiples capas, todo de una manera fiable y sencilla.

Delphi 4

Esta versión de Delphi se enfocó en la facilidad de desarrollo y en el cómputo distribuido a través de MIDAS (Servicios de Desarrollo de Aplicaciones de Múltiples de Capas), DCOM y CORBA. En cuanto al IDE aportó capacidades de auto-completar el código fuente y de navegación de código, el navegador de módulos presentó además mejoras en su depurador integrado.

Delphi 5

Delphi 5 agregó mejoras al IDE, como la configuración de diferentes escritorios, nuevos editores de propiedades en el inspector de Objetos, Diagramas de Datos, Listas de Pendientes, además de soporte para control de versiones a través de TeamSorce, soporte para ADO y conexión a Interbase sin BDE.

Delphi 6

Con el desarrollo interplataforma Delphi 6 representó una mejora sin precedentes, al proporcionar compatibilidad con Kylix para desarrollo linux a través de la CLX (Component Library for Cross Platform), además incorporó cambios al IDE como el Object Tree View, pestañas de vista múltiple en el editor de código, mejoras al inspector de Objetos, nuevos esquemas de conexión a bases de datos como DBExpress, soporte para XML, además de mejoras al compilador como compilación condicional a través de constantes definidas.

Delphi 7

Delphi 7 incluye los nuevos componentes de IntraWeb para desarrollo web, nuevos drivers de DBExpress para Informix SE, Oracle 9i, DB2 7.2, nuevos componentes Rave para generación de reportes, soporte para modelado con UML a través de ModelMaker y de los componentes Bold, además de mejoras al IDE y soporte para desarrollar aplicaciones .NET.

Delphi 8

La más reciente versión incluye novedades que se centran, sobre todo, en su capacidad para crear rápidamente aplicaciones para la Web, pero también incorpora una amplia compatibilidad con Kylix y todos los nuevos elementos propios de las últimas GUIs de Windows.

1.3 Ediciones de Delphi

- “Studio Architect Edition
  Architect Edition para el diseño y desarrollo de aplicaciones orientadas a modelos
- Studio Enterprise Edition
  Para equipos que desarrollan aplicaciones de bases de datos de clase empresarial
- Studio Professional Edition

Professional Edition, ideal para individuos que desarrollan aplicaciones para la Web y GUI para la infraestructura Microsoft .NET.

- Personal Edition

Los requerimientos de instalación de cada edición son los siguientes:

Studio Architect Edition

- Intel Pentium 233 MHz o superior
- Microsoft Windows XP, Windows 2000, o Windows 98
- 64MB RAM (128Mb recomendados)
- 124MB de espacio en disco duro (instalación compacta)
- 520MB de espacio en disco duro (instalación completa)
- Unidad de CD-ROM
- SVGA o monitor de alta resolución
- Mouse u otro dispositivo apuntador

Studio Enterprise Edition

- Intel Pentium 233 MHz o superior
- Microsoft Windows XP, Windows 2000, o Windows 98
- 64MB RAM (128Mb recomendados)
- 124MB de espacio en disco duro (instalación compacta)
- 450MB de espacio en disco duro (instalación completa)
- Unidad de CD-ROM
- SVGA o monitor de alta resolución
- Mouse u otro dispositivo apuntador

Studio Professional Edition

- Intel Pentium 233 MHz o superior
- Microsoft Windows XP, Windows 2000, o Windows 98
- 64MB RAM (128Mb recomendados)
- 110MB de espacio en disco duro (instalación compacta)
- 400MB de espacio en disco duro (instalación completa)
- Unidad de CD-ROM
- SVGA o monitor de alta resolución
- Mouse u otro dispositivo apuntador

Personal Edition

- Intel Pentium 233 MHz o superior
- Microsoft Windows XP, Windows 2000, o Windows 98
- 32MB RAM (128Mb recomendados)
- 75MB de espacio en disco duro (instalación compacta)
- 160MB de espacio en disco duro (instalación completa)
- Unidad de CD-ROM
- SVGA o monitor de alta resolución
- Mouse u otro dispositivo apuntador"[2]

Las características más importantes de cada edición se presentan en la siguiente tabla:

	Studio Architect	Studio Enterprise	Studio Professional	Personal Edition
Nueva Interoperabilidad y soporte a la migración a Microsoft .NET				
Advertencias de compilación de compatibilidad a .NET	•	•	•	•
Permite importar cualquier ensamblaje de .NET como un objeto COM.	•	•	•	•
Exportar objetos COM de Delphi a aplicaciones manejadas de .NET	•	•	•	•
Nueva versión preliminar de Delphi para .NET				
Compilador de CIL de preliberacion para .NET	•	•	•	•
Documentación de migración para .NET	•	•	•	•
Nueva Tecnología de ModelMaker				
Diseño basado en modelado	•	•		
Modelado visual basado en UML y reingeniería.	•	•		
Integración Nativa en Delphi, ingeniería inversa y visualización instantánea	•	•		
Aplicación de Patrones integrada.	•	•		
Nueva tecnología intraWeb de AtoZed				
Generación de aplicaciones para Internet utilizando desarrollo visual.	•	•	•	
Añade rápidamente contenido interactivo a su sitio Web, así como construcción dinámica de aplicaciones Web.	•	•	•	
Manejo transparente de los detalles de las aplicaciones Web como cookies, sesiones y administración de usuarios.	•	•		
Creación, depuración y mantenimiento de aplicaciones basadas en Web fácilmente con el entorno visual RAD.	•	•		

[2] IDEM

Soporte para utilización sencilla de API's para implementar componentes personalizados.	•	•		
Nueva tecnología Rave de Nevrona				
Poderoso diseñador visual de reportes Rave Reports Borland Edition y código basado en API's	•	•	•	
Formatos de visualización en PDF, HTML, RTF y texto.	•	•	•	
Soporte nativo para aplicaciones de VCL y CLX.	•	•	•	
Diseño flexible basado en páginas para manejar estilos de reportes de forma y bandas.	•	•	•	
Nuevos temas de Windows XP				
Aplicaciones que toman parte de los temas de Windows XP.	•	•	•	•
Apariencia de Windows XP a través de la librería de controles comunes.	•	•	•	•
API´s de ThemeServices que permiten a una aplicación utilizar las funciones de temas transparentemente.	•	•	•	•
BizSnap – Servicios Web con tecnología XML				
Navegador UDDI para localizar visualmente e importar Servicios Web registrados.	•	•	•	
Soporte de manejo de fallos UDDI de cliente.	•	•	•	
Arquitectura global XML para servicios Web e inspección de lenguaje de Servicios Web.	•	•	•	
Desarrollo rápido de servicios Web del lado de servidor conforme a W3C utilizando SOAP, XML, WSDL y más.	•	•	•	
Desarrollo rápido de aplicaciones cliente que consuman servicios Web conforme a W3C utilizando SOAP, XML, WSDL y más.	•	•	•	
Asistentes de WSDL y componentes para acceder fácilmente a servicios Web en Internet y agregar funcionalidad a sus propias aplicaciones.	•	•	•	
Creación de aplicaciones que operen transparentemente a través de Internet con servicios Web basados en estándares en plataformas como Microsoft, .NET, ONE de Sun y otros.	•	•	•	
Asistentes de comunicación SOAP y componentes para construir Servicios Web rápidamente.	•	•	•	
Objeto de documento XML en la VCL que proporciona acceso rápido a nuevos formatos XML.	•	•		
Objetos de documento XML y servicios Web WSDL nativos que interactúan con codeinsight y la verificación de tipos en tiempo de compilación.	•	•		

WebSnap – La plataforma de diseño de aplicaciones Web completa				
Distribuye sus aplicaciones Web a Microsoft IIS, Netscape y Apache 2.0 con soporte a ISAPI, NSAPI, Apache DSO y CGI.	•	•		
Diseñadores de superficie de páginas web para construir rápidamente y visualizar las superficies de aplicaciones WebSnap en HTML e instantáneamente actualizar la vista previa del navegador.	•	•		
Arboles de XML y XSL para representar documentos XML y XSL con identaciones para facilitar la lectura.	•	•		
Múltiples Web Modules en un solo proyecto para organizar las páginas del sitio.	•	•		
Page Producers de XLS para combinar fuentes de datos XML con transformaciones XSL para producir salidas a HTML.	•	•		
Soporte para depuración de Scripts Activos en WebSnap.	•	•		
InternetExpress				
Tablas XML de Navegador Web para un alto desempeño de cache de cliente.	•	•		
Datos XML de servidores de DataSnap para simplificar el intercambio de información.	•	•		
Editor de páginas Web para diseñar instantáneamente documentos Web de HTML 4 que proporcionen datos dinámicos de XML.	•	•		
Asistentes y Componentes de Internet / Intranet				
WebBridge como una solución abierta que soporta ISAPI, NSAPI, DLLs de Apache y CGI.	•	•	•	
Vista previa de páginas con soporte HTML 4.	•	•	•	
Asistentes de distribución Web para distribuir clientes delgados, aplicaciones con cero – configuración utilizando el Web.	•	•	•	
ActiveForms para construir formas GUI para aplicaciones cliente de Win32 y navegadores.	•	•	•	
Archivos CAB de BDE para distribución sencillas de aplicaciones de bases de datos sobre el Web.	•	•	•	
Soporte para JPG	•	•	•	
Asistente de Objetos de Servidor Activo para desarrollo de servidores ASP de alto desempeño.	•	•	•	
DataSnap				
Distribución libre de cargos con licencias ilimitadas.	•	•		
Clientes Web, clientes GUI y acceso a servicios Web a cualquier RDBMS soportado.	•	•		
Conexiones de acceso a SOAP / XML, COM, Web y TCP/IP disponibles para una máxima conectividad de redes y flexibilidad	•	•		

Construcción sencilla de interfaces a servicios Web SOAP/XML a Oracle, SQL-Server, DB2, Interbase y más.	•	•		
Arquitectura basada en conjuntos de datos para una curva rápida de aprendizaje.	•	•		
Soporte para las arquitecturas de acceso de datos BDE, IBX y dbExpress.	•	•		
Propagación automática de restricciones de bases de datos para proporcionar las reglas de negocios a las aplicaciones cliente para procesamiento local.	•	•		
Aplicaciones delgadas del lado del cliente, de bajo mantenimiento y fácil configuración para reducir los costos de desarrollo.	•	•		
Licencia de desarrollo de DataSnap incluida.	•	•		
Proporciona información a aplicaciones de clientes delgados de forma rápida, eficiente y segura.	•	•		
Soporte para Corba				
Visibroker 4.5 para Delphi 7 incluyendo desarrollo de clientes y servidores CORBA.	•	•		
Asistentes para simplificar el desarrollo de clientes y servidores CORBA.	•	•		
Soporte para Borland Enterprise Server, Edición AppServer SIDL.	•	•		
Soporte simultáneo para objetos COM y CORBA.	•	•		
Aplicaciones nativas de Windows de alto desempeño				
Compilador de código nativo de 32 bits optimizado de alto rendimiento.	•	•	•	•
Fácil creación de librerías de ligado dinámico (DLLs), controles COM y ejecutables independientes.	•	•	•	•
Ensamblador integrado con soporte para el conjunto completo de instrucciones Intel de 32 bits (Incluyendo Pentium Pro, Pentium IV, MMX, SIMD y AMD 3Dnow).	•	•	•	•
Soporte para números complejos vía variants personalizados	•	•	•	•
Directiva de alerta deprecated para mejorar el desarrollo de aplicaciones.	•	•	•	•
Directivas de alerta de plataforma y librerías para asistir en el desarrollo interplataforma.	•	•	•	•
Desarrollo Rápido de Aplicaciones (RAD)				
IDE con editor de código integrado, depurador, vista histórica, mensajes de error de fácil compresión y resalte de sintaxis por colores.	•	•	•	•
Extensión de resalte de sintaxis con OpenTools API con soporte para JavaScript, PHP y archivos INI.	•	•	•	•
Listas de pendientes para mantener el desarrollo a tiempo.	•	•	•	•

Navegador de proyectos para comprender mejor el código y navegar por la VCL.	•	•	•	
Asistente del panel de control para construir applets de control de manera rápida y simple.	•	•	•	
Editor de mapeo de teclas para afinar el editor a la manera que usted trabaje.	•	•	•	
Máxima Reutilización con la Arquitectura Orientada a Objetos de Delphi				
Arquitectura de Aplicaciones y Componentes Orientada a Objetos completamente extensibles tanto para VCL y CLX.	•	•	•	•
Soporte para controles comunes de Windows	•	•	•	•
Herencia Visual de Formas en VCL y CLX y ligado de formas para reducir la codificación y simplificar el mantenimiento.	•	•	•	•
Repositorio de Objetos para almacenar y reutilizar formas, DataModules y Expertos.	•	•	•	•
Biblioteca de componentes visuales para arrastrar y soltar componentes reutilizables.	•	•	•	•
Biblioteca de componentes para desarrollo interplataforma para arrastrar y soltar componentes reutilizables para Windows y Linux.	•	•	•	
Explorador de código para una referencia instantáneamente actualizada de los archivos de unidades de una aplicación.	•	•	•	
CodeInsight – Acelera la codificación y Reduce los Errores de Sintaxis				
Asistente de plantillas de código para simplificar la creación de código.	•	•	•	•
Plantillas de código mejoradas con la habilidad de crear, modificar y exportar plantillas para cualquier tipo de archivo.	•	•	•	•
Asistente de conclusión de código para asegurar una sintaxis correcta.	•	•	•	•
Asistente de parámetros de código para desplegar la lista de parámetros de procedimientos, métodos y eventos.	•	•	•	•
Evaluación de Expresiones de ToolTip para una depuración más fácil.	•	•	•	•
Atajos de teclado de navegación de clases.	•	•	•	•
Conclusión de Clases.	•	•	•	
Evaluación de Símbolos de ToolTip.	•	•	•	
Depurador Avanzado				
Ventana de watch de múltiples ventanas para agrupar variables lógicamente	•	•	•	•
Nombrado de Threads para depurar fácilmente aplicaciones multi-threads, con nombres definidos por el usuario en lugar de definidos por ID de thread.	•	•	•	•
Depuración de DLLs para mejorar el control de las extensiones de aplicación.	•	•	•	•
Vista de Módulos.	•	•	•	•
Ventana de CPU para depuración e bajo nivel.	•	•	•	•

Winsight 32 para monitorear mensajes de Windows.	•	•	•	
Debug Inspector para monitorear propiedades de componentes de depuración.	•	•	•	
Ventana de variables locales.	•	•	•	
Depurador de procesos remotos para desarrollo distribuido.	•	•		
TeamSource para escalar el RAD al equipo completo de desarrollo				
Mantenga la productividad de desarrollo mientras protege su código fuente.	•	•		
Administre fácilmente el código fuente en equipos grandes de desarrollo distribuido.	•	•		
Controle los cambios realizados a los archivos maestros con versiones históricas.	•	•		
Simplifique el manejo de los hitos de su proyecto con marcadores de código fuente.	•	•		
Soporte para PVCS y otros motores de control de versiones con controladores de versiones modulares.	•	•		
Biblioteca de Componentes Visuales				
Componentes nativos de VCL para desarrollo rápido de aplicaciones.	300+	300+	225+	85+
Soporte para Windows XP Themes para modernizar la apariencia de sus aplicaciones.	•	•	•	•
Clases de Subcomponentes para combinar elementos utilizados frecuentemente para acelerar la creación de interfaces de usuario.	•	•	•	•
La unidad conversión simplifica las conversiones de medidas.	•	•	•	•
Administrador de paquetes para controlar fácilmente el contenido de paquetes personalizados de componentes.	•	•	•	•
Action List para manejar código frecuentemente utilizado en una aplicación.	•	•	•	•
Editor de colecciones de paquetes.	•	•	•	
Componentes de Navegador Web para integrar navegación de HTML en su aplicación.	•	•	•	
Utilice componentes de automatización de Microsoft Office para integrar rápidamente sus aplicaciones con aplicaciones de Office como Word, Excel y Outlook.	•	•	•	
Desarrollo de CLX para Windows/Linux				
Más de 160 componentes nativos de CLX para desarrollo rápido de aplicaciones en Windows/Linux.	•	•	•	
Clases y componentes de BaseCLX RTL.	•	•	•	
Componentes nativos de GUI Visual CLX y controles visuales data-aware.	•	•	•	
Diálogos comunes de GUI CLX-Abrir, Salvar, Fuente, Buscar Y Reemplazar.	•	•	•	
Componentes altamente escalables de acceso a datos DataCLX.	•	•	•	

Borland Kylix 3 para el lenguaje Delphi				
IDE de Borland Kylix Enterprise para el Lenguaje Delphi.	•	•		
IDE de Borland Kylix Professional para el lenguaje Delphi.			•	
Código Fuente de Borland				
Código Fuente de VCL.	•	•	•	
Código Fuente de CLX.	•	•	•	
Código Fuente de Editores de Propiedades.	•	•	•	
Un amplio rango de opciones de acceso a bases de datos				
DbExpress para acceso nativo SQL RDBMS de alto rendimiento.	•	•	•	
Borland Database Engine (BDE) 5.1.1.	•	•	•	
Interbase Express (IBX) para acceso directo por APIs a Interbase.	•	•	•	
Arquitectura abierta de base de datos				
Drivers de Access, Foxpro, Paradox y dBase para acceso de alta velocidad a sistemas de bases de datos de escritorio y LAN.	•	•	•	
Conectividad ODBC completa.	•	•	•	
API de BDE para acceso de abierto a cualquier motor de base de datos.	•	•	•	
Soporte para refrescado automático.	•	•	•	
Clase TcustomConnection para integrar fácilmente soluciones de bases de datos de terceros.	•	•	•	
Drivers nativos de SQL links con licencia de distribución ilimitada para Interbase, Oracle, Sybase, Informix, SQL Server y DB2.	•	•		
Drivers de dbExpress para MS SQL 2000.	•	•		
Drivers de dbExpress para Interbase, MySQL.	•	•	•	
Drivers locales de BDE para paradox, dbase, Foxpro, Access.	•	•	•	
Drivers de dbExpress para Oracle, DB2 e Informix.	•	•		
SQL Links de BDE para Oracle, SQL Server, DB2, Informix, Sybase e Interbase.	•	•		

1.4 Preguntas de Repaso

1. Defina con sus propias palabras ¿Qué es Delphi?

2. Antes de su liberación del Delphi 7 ¿Cómo se llamaba?

a) ”Alejandrina” b)”Aurora” c)” AppBuilder”

3. ¿Cuántas ediciones existen de Delphi 7?

4. ¿Cuales son los nuevos componentes que agregó Delphi 7?

a) IntraWeb, Oracle9i y Rave b) IDE, Oracle9i y Diag. de Datos c) Rave, MIDAS y CORBA

5. Mencione 5 características de Delphi.

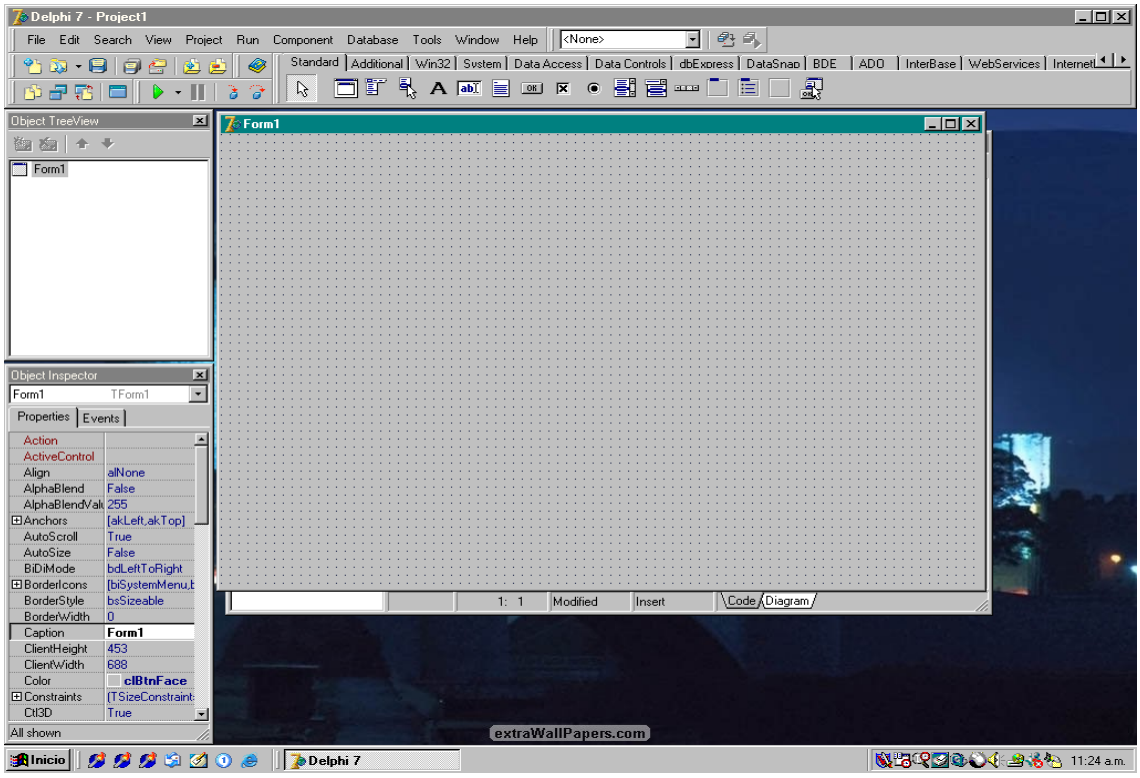
6. ¿Qué ayuda ofrece la nueva tecnología Rave de Nevrona?

7. ¿Con cuantos componentes nativos cuenta el Studio Architect?

II. Ambiente Integrado de Desarrollo

"Delphi proporciona un Ambiente Integrado de Desarrollo (IDE. Integrated Development Environment) que facilita el desarrollo al proporcionar las herramientas necesarias para el diseñar, ejecutar y depurar sus aplicaciones. Este ambiente está diseñado de una manera organizada y funcional, aunque lo puede adaptar a su gusto personal."[1]

2.1 La ventana Principal



La ventana principal de Delphi esta dividida en las siguientes secciones:

- **El menú Principal.**
- **La barra de Herramientas.**
- **La paleta de Componentes.**
- **La forma.**
- **El Árbol de Objetos.**
- **El inspector de Objetos.**

[1] http://www.borland.com/delphi/

- **El editor de Código.**
- **El Explorador de Código.**
- **La Ventana de Diagrama.**

A continuación describiremos cada una de estas secciones.

2.2 El Menú Principal

A través del menú de Delphi se puede realizar diversas acciones las cuales están agrupadas de la siguiente forma:

File

Por medio del menú File podemos invocar acciones relacionadas con el proyecto actual y sus archivos, estas opciones se describen a continuación:

SUBMENU	DESCRIPCIÓN
New	Permite crear distintos tipos de proyectos o elementos de una aplicación.
Open	Permite abrir un proyecto existente o distintos tipos de archivos.
Open project Ctrl+F11	Permite abrir un proyecto o un grupo de proyectos existentes.
Reopen	Permite abrir los proyectos, grupos de proyectos o archivos abiertos recientemente.
Save Ctrl+S	Salva el archivo actual.
Save As	Permite salvar el archivo actual con un nombre distinto.
Save Project As	Permite salvar el proyecto actual con un nombre distinto.
Save All Shift+Ctrl+S	Salva todos los archivos del proyecto actual.
Close	Cierra el archivo actual.
Close All	Cierra el proyecto actual.
Use Unit Alt+F11	Permite utilizar una unidad lógica de código desde otra unidad.
Print	Imprime el archivo actual.
Exit	Cierra Delphi.

Edit

Las opciones del menú Edit se describen a continuación:

SUBMENÚ	DESCRIPCIÓN
Undo Ctrl+Z	Permite deshacer la última acción ejecutada.
Redo Shift+Ctrl+Z	Rehace la última acción deshecha.
Cut Ctrl+X	Corta el elemento seleccionado al portapapeles.
Copy Ctrl+C	Copia el elemento seleccionado al portapapeles.
Paste Ctrl+V	Pega el contenido del portapapeles.
Delete Ctrl+Del	Elimina el elemento seleccionado.
Select All Ctrl+A	Selecciona todo el contenido.
Align to Grid	Permite alinear un componente dentro del diseñador de forma.
Bring to Front	Coloca un control al frente de otros controles.
Send to Back	Coloca un control detrás de otros controles.
Align	Permite alinear los controles de una forma.
Size	Permite modificar las dimensiones de un control.
Scale	Permite escalar las dimensiones de un control.
Tab Order	Permite modificar el orden de tabulación de los controles de un contenedor.
Lock Controls	Bloquea los controles de una forma para impedir que se puedan mover o redimensionar.
Add to Interface	Agrega un elemento a una interfaz.

Search

La siguiente tabla muestra las opciones del menú Search:

SUBMENÚ	DESCRIPCIÓN
Find Ctrl+F	Permite buscar una cadena en la unidad actual o en las unidades de un proyecto.
Find in Files	Permite buscar una cadena en las unidades de un proyecto.
Replace Ctrl+R	Permite substituir una cadena de texto por otra.
Search Again F3	Realiza nuevamente la búsqueda anterior.
Incremental Search Ctrl+E	Realiza una búsqueda incremental sobre la unidad actual.
Go to Line Number Alt+G	Posiciona el cursor en una línea especifica de la unidad actual.
Find Error	Permite buscar un error en el proyecto actual.
Browse Symbol	Permite buscar un símbolo.

View

Las opciones del menú View se describen a continuación:

SUBMENÚ	DESCRIPCIÓN
Project Manager Ctrl+Alt+F11	Muestra la ventana del administrador de Proyectos.
Translation Manager	Muestra la ventana del Administrador de Traducciones.
Object Inspector F11	Muestra la ventana del Inspector de Objetos.
Object TreeView Shift+Alt+F11	Muestra la ventana del árbol de Objetos.
To-Do List	Muestra la ventana de pendientes.
Alingnment Palette	Muestra la Paleta de alineación.
Browser Shift+Ctrl+B	Muestra la ventana del navegador de clases.
Code Explorer	Muestra el navegador de código.
Component List	Muestra una ventana para seleccionar componentes para agregarlos a una forma.
Window List Alt+0	Muestra la lista de todas las ventanas abiertas.
Additional Message Info	Despliega la ventana de Tips de Mensajes.
Debug Windows	Muestra las ventanas de depuración.
Desktops	Permite acceder a las opciones de escritorios.
Toggle Form/Unit F12	Permite conmutar entre el editor de código y el diseñador de formas.
Units Ctrl+F12	Muestra las unidades del proyecto actual.
Forms Shits+F12	Muestra las formas del proyecto actual.
Type Library	Permite examinar y crear información de tipos para controles Actives.
New Edit Window	Permite crear una nueva ventana del editor de código.
Toolbars	Permite personalizar las barras de herramientas.

Project

Las opciones del menú Project son las siguientes:

SUBMENÚ	DESCRIPCIÓN
Add to Project Shift+F11	Permite agregar un archivo al proyecto actual.

Remove from Project	Permite remover archivos del proyecto actual.
Import Type library	Permite agregar proyectos, unidades o formas al repositorio de objetos.
View Source	Muestra el código fuente del archivo .DPR del proyecto.
Languages	Permite modificar las opciones de traducción del proyecto.
Add New Project	Permite agregar un proyecto nuevo a un grupo de proyectos.
Add Existing Project	Permite agregar un proyecto existente a un grupo de proyectos.
Compile [Proyecto] Ctrl+F9	Compila el proyecto actual.
Build [Proyecto]	Reconstruye el proyecto actual.
Syntax check [Proyecto]	Permite compilar los módulos de compilación del proyecto actual.
Information for [Proyecto]	Muestra la información de compilación del proyecto actual.
Compile All Projects	Reconstruye todos los proyectos de un grupo de proyectos.
Web Deployment Options	Permite configurar un control ActiveX o una ActiveForm para liberarla a un servidor Web.
Web Deploy	Libera un control ActiveX o una ActiveForm en un servidor Web.
Options Shift+Ctrl+F11	Muestra la ventana de opciones del proyecto actual.

Run

Las opciones del menú Run se describen a continuación:

SUBMENÚ	DESCRIPCIÓN
Run F9	Permite ejecutar el proyecto actual.
Attach to Process	Permite depurar un proyecto que está actualmente en ejecución.
Parameters	Permite configurar los parámetros del proyecto actual.
Register ActiveX Server	Permite registrar un servidor Actives.
Unregister ActiveX Server	Permite eliminar el registro de un servidor Actives.
Install COM+Objects	Permite ejecutar los objetos de una aplicación baja COM+.
Step Over F8	Ejecuta un programa una línea a la vez, ejecutando procedimientos como una sola unidad.
Trace Into F7	Ejecuta un programa una línea a la vez, ejecutando procedimientos una línea a la vez.
Trace to Next Source Line	Detiene la ejecución de un programa en la siguiente línea de

Shift+F7	código.
Run to Cursor F4	Ejecuta el proyecto actual hasta la línea de código donde se encuentre el cursor en el Editor de código.
Run Until Return Shift+F8	Ejecuta el programa hasta que la ejecución regresa de la función actual.
Show Execution Point	Posiciona el cursor en el punto de ejecución actual del programa en la ventana del Editor de código.
Program Pause	Pausa temporal de la ejecución del proyecto actual.
Program Reset Ctrl+F2	Termina la ejecución del programa y lo libera de la memoria.
Inspect	Proporciona información del elemento seleccionado actualmente.
Evaluate/Modify Ctrl+F7	Permite evaluar o modificar el valor actual de una expresión.
Add Watch Ctrl+F5	Permite crear o modificar un elemento en la ventana Watches.
Add Breakpoint	Permite agregar un punto de ruptura al proyecto actual.

Component

La siguiente tabla muestra las opciones del menú Component:

SUBMENÚ	DESCRIPCIÓN
New Component	Permite crear un nuevo componente.
Install Component	Permite instalar un nuevo componente en la paleta de componentes.
Import ActiveX Control	Permite agregar un control ActiveX registrado a la paleta de componentes.
Create component Template	Permite crear una plantilla de componentes.
Install Packages	Permite configurar los paquetes instalados en Delphi.
Confige Palette	Permite configurar la paleta de componentes.

Database

Las opciones del menú Database se describen a continuación:

SUBMENÚ	DESCRIPCIÓN
Explore	Ejecuta el SQL Explorer para poder examinar una base de datos.

SQL Monitor	Ejecuta el SQL Monitor para examinar las llamadas realizadas a un servidor remoto a través de SQL Links o un socket ODBC.
Form Wizard	Permite crear una forma con componentes de acceso a datos.

Tools

Las siguientes opciones del menú Tools son:

SUBMENÚ	DESCRIPCIÓN
Environment Options	Permite personalizar las opciones de configuración del IDE.
Editor Options	Permite personalizar la configuración del Editor de código.
Debugger Options	Permite personalizar la configuración del depurador.
Translation Tools Options	Permite configurar las herramientas de traducción.
Repository	Permite configurar el repositorio de objetos.
Translation Repository	Permite almacenar y recuperar cadenas del administrador de traducciones.
Web App Debugger	Ejecuta el Web App Debugger.
Regenerate CORBA IDL Files	Genera una aplicación cliente o servidora utilizando un archivo basado en IDL.
Configure Tools	Permite agregar, eliminar o editar programas en el menú Tools.
Database Desktop	Ejecuta el database desktop.
Image Editor	Ejecuta el image editor.
Package Collection Editor	Permite crear una colección de paquetes para distribuirlos.
XML Mapper	Permite definir mapeos entre documentos XML y paquetes de datos.
Rave Designer	Permite crear reportes Rave.

Window

Las opciones del menú Window de describen a continuación:

SUBMENÚ	DESCRIPCIÓN
Object Inspector	Muestra la ventana del inspector de objetos.
Object TreeView	Muestra la ventana del árbol de objetos.
Next Window Alt+End	Coloca el foco en la siguiente ventana.

Help

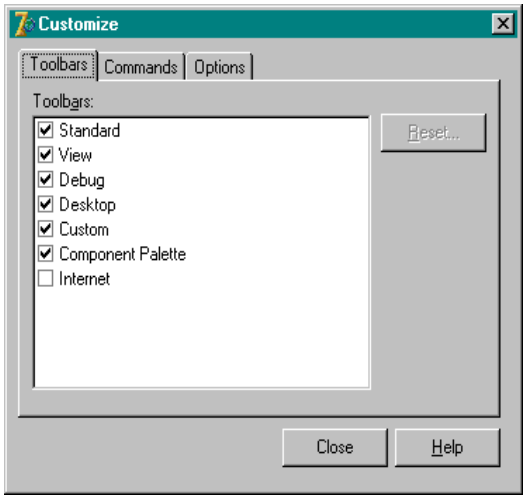
Las opciones del menú Help se describen en la siguiente tabla:

SUBMENÚ	DESCRIPCIÓN
Delphi Help	Invoca la ayuda de Delphi.
Delphi Tools	Invoca la ayuda de las herramientas de Delphi.
Windows SDK	Invoca la ayuda del kit de desarrollo de Windows.
Borland Home Page	Abre la página principal de Borland.
Borland Developer Network	Abre la página del Borland Developer Network.
Delphi Home Page	Abre la página principal de Delphi.
Delphi Developer Support	Abre la página de soporte de Delphi.
Delphi Direct	Abre la ventana de Delphi Direct.
Customize	Permite configurar los archivos de ayuda de Delphi.
About	Muestra la ventana de Acerca de…

2.3 La Barra de Herramientas

La barra de Herramientas de Delphi se muestra a continuación:

La barra de Herramientas posee un conjunto de botones de acceso rápido a acciones de uso común del menú de Delphi. Se puede personalizar pulsando click con el botón derecho del Mouse sobre ésta y seleccionando la opción Customize del menú contextual, que se muestra en el siguiente cuadro de diálogo:



Por medio de la pestaña Toolbars, del cuadro de diálogo Customize, podemos seleccionar las barras de herramientas que se desea visualizar. Estas son:

- **Standard**
- **View**
- **Debug**
- **Desktop**
- **Custom**
- **Component Palette**
- **Internet**

A continuación se describen los elementos de cada una de estas barras de herramientas:

Standard

New Items. Despliega el cuadro de diálogo New Items.

Open. Permite abrir un proyecto o unidad abierto recientemente.

 Save. Salva el archivo actual.

 Save All. Salva todos los archivos del proyecto actual.

 Open Project. Abre un proyecto existente.

 Add file to Project. Permite agregar un archivo al proyecto actual.

 Remove file from Project. Permite eliminar un archivo del proyecto actual.

View

 View Unit. Muestra las unidades del proyecto.

 View Form. Muestra las formas del proyecto.

 Toggle Form/Unit. Conmuta entre una forma y su unidad de código.

 New Form. Crea una nueva forma en el proyecto.

Debug

 Run. Ejecuta el proyecto actual.

 Pause. Pausa la ejecución del proyecto actual.

 Trace into. Ejecuta un programa una línea a la vez, ejecutando procedimientos una línea a la vez.

 Step over. Ejecuta un programa una línea a la vez, ejecutando procedimientos como una sola unidad.

Desktop

Este ComboBox permite seleccionar entre distintos escritorios previamente salvados.

 Save current desktop. Salva la configuración actual del escritorio.

 Set debug desktop. Permite seleccionar entre los distintos escritorios salvados.

Custom

Esta barra de herramientas está diseñada para personalizarla agregando nuevos botones.

 Help contents. Muestra la ayuda de Delphi.

Component Palette

Muestra u oculta la Paleta de Componentes.

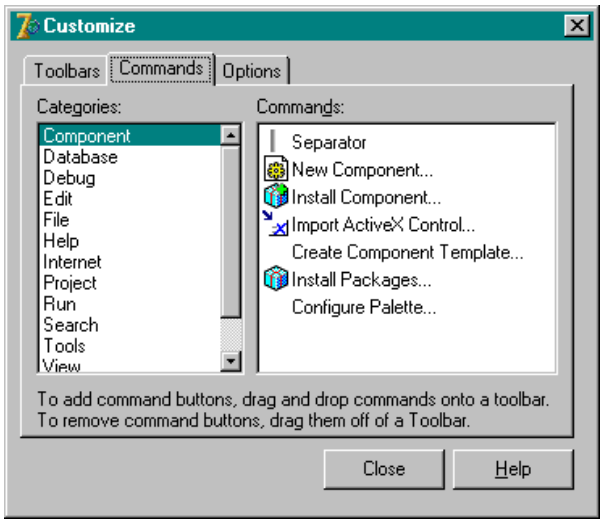
Internet

 New WebSnap Data Module. Crea un Data Module de WebSnap.

 New WebSnap Page Module. Crea una Página de WebSnap.

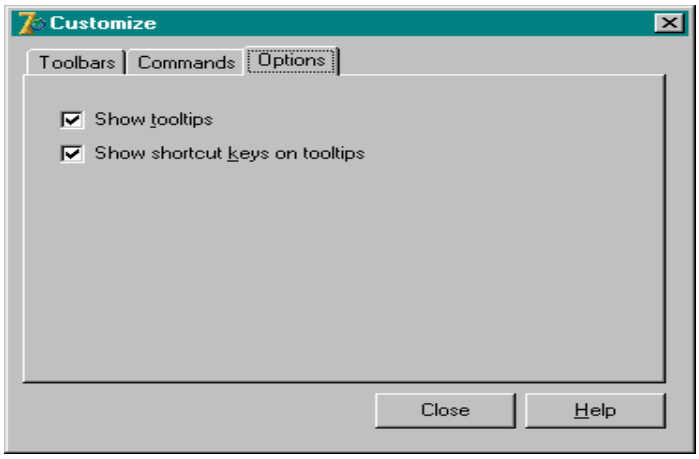
 New WebSnap Application. Crea una nueva aplicación de WebSnap.

Adicionalmente las barras de herramientas pueden personalizarse a través de la pestaña Commands del cuadro de diálogo Customize, la cual se muestra a continuación.



Se puede seleccionar un comando en la sección Commands y arrastrarlo hasta una de las barras de herramientas, similarmente, si se desea eliminar un botón de una barra de herramientas se debe arrastrar el botón fuera de la barra para eliminarlo.

La pestaña Options de este cuadro de diálogo permite configurar, si se desea, que al pasar el cursor del Mouse sobre los botones de las barras de herramientas se muestre una leyenda descriptiva de la función que realiza el botón, junto con las teclas de acceso rápido (si existen) que ejecuten la acción del botón, tal como se muestra en la siguiente pantalla.



2.4 La Paleta de Componentes

Delphi permite utilizar componentes para construir nuestras aplicaciones. Un componente es un objeto que permite utilizar una y otra vez cierta funcionalidad, facilitando así el desarrollo de una aplicación.

"La Paleta de componentes permite agregar un componente a una forma, los componentes que la integran pueden ser visuales en tiempo de ejecución o no. La Paleta de Componentes está dividida en pestañas, cada una de las cuales agrupa un conjunto de componentes que proporcionan una funcionalidad o comportamiento similar; se pueden agregar nuevos componentes a la Paleta ya sea en pestañas existentes o en nuevas pestañas, de esta forma se agrega nueva funcionalidad a las aplicaciones de Delphi."

Si una pestaña de la Paleta de Componentes contiene más componentes de los que se pueden mostrar en pantalla, en el extremo derecho de la pestaña se ubica un botón con una doble flecha que permite el acceso a los componentes restantes. Tal como se muestra enseguida.

A continuación se describen las pestañas de la Paleta de Componentes.

Standard

Los componentes en la pestaña Estándar representan los controles estándar de Windows.

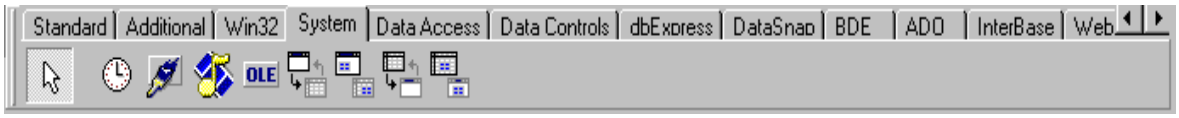
Additional

Los componentes en la pestaña Additional representan controles especializados de Windows disponibles para las aplicaciones.

Win32

Los componentes en la pestaña Win32 representan controles de interfaz de Windows de 32 bits.

System



Los componentes en la pestaña System representan controles de sistema especializados disponibles para las aplicaciones.

Data Access

Los componentes en la pestaña Data Access permiten trabajar con la información de una base de datos. Estos componentes no están asociados con un mecanismo de acceso a datos en particular.

Data Controls

Los componentes en la pestaña Data Controls representan controles especializados para mostrar y manipular la información de una base de datos.

DbExpress

Los componentes en la pestaña dbExpress permiten a las aplicaciones comunicarse con bases de datos utilizando dbExpress.

DataSnap

Los componentes en la pestaña DataSnap permiten construir aplicaciones de bases de datos de múltiples capas.

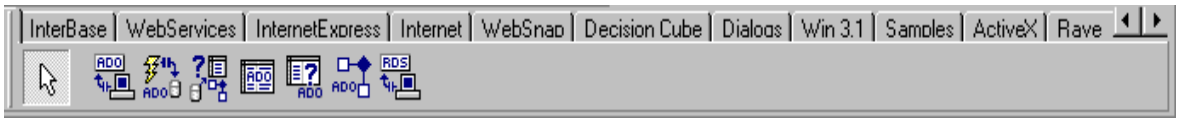
BDE

Los componentes en la pestaña BDE permiten conectarse a la información de una base de datos utilizando el Borland Database Engine (BDE).

ADO

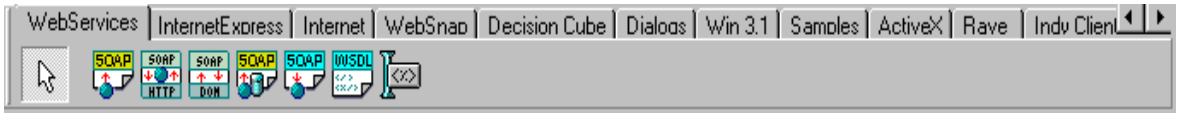
Los componentes de la pestaña ADO permiten conectarse a la información de una base de datos utilizando Objetos de datos ActiveX (ADO).

Interbase



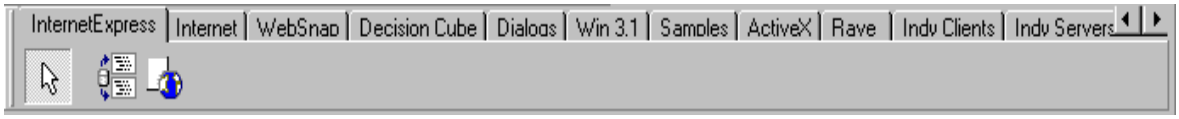
Los componentes en la pestaña Interbase permiten conectarse directamente a una base de datos de Interbase sin necesidad de utilizar un motor de acceso a datos como BDE o ADO.

WebServices



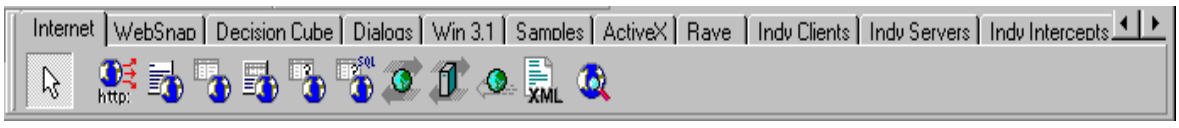
Los componentes en la pestaña WebServices permiten escribir aplicaciones cliente que accedan a WebServices a través de SOAP.

InternetExpress



Los componentes en la pestaña InternetExpress permiten construir aplicaciones de InternetExpress que son simultáneamente una aplicación de Servidor Web y el cliente de una aplicación de base de datos de múltiples capas.

Internet



Los componentes en la pestaña Internet permiten crear aplicaciones de Servidor Web.

WebSnap

Los componentes en la pestaña WebSnap permiten crear aplicaciones de Servidor Web que contengan páginas Web complejas manejadas por eventos.

Decision Cube

Los componentes en la pestaña Decision Cube agregan capacidad de análisis de datos multidimensionales a las aplicaciones.

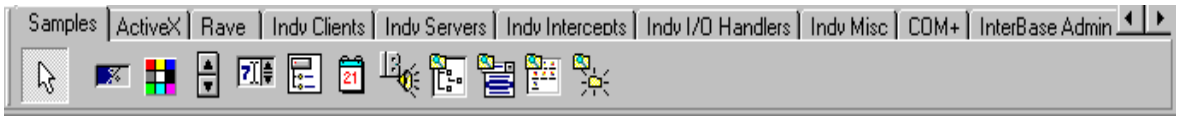
Dialogs

Los componentes en la pestaña Dialogs de Paleta de componentes constituyen los cuadros de diálogo estándar de Windows.

Win 3.1

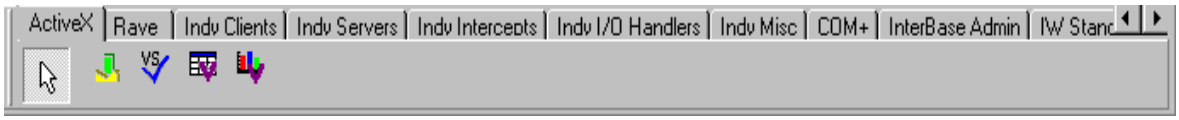
Los componentes en la pestaña Win 3.1 representan controles elementales de Windows 3.1 proporcionando compatibilidad con aplicaciones construidas con versiones anteriores de Delphi, no deberán utilizarse en el desarrollo de nuevas aplicaciones.

Samples



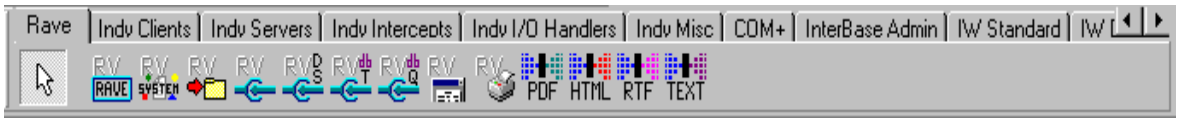
Los componentes en la pestaña Samples representan ejemplos de componentes personalizados que se pueden desarrollar e instalar en la paleta de componentes.

ActiveX



Los componentes en la pestaña ActiveX representan objetos ActiveX.

Rave



Los componentes en la pestaña Rave permiten generar reportes Rave de Nevrona en las aplicaciones.

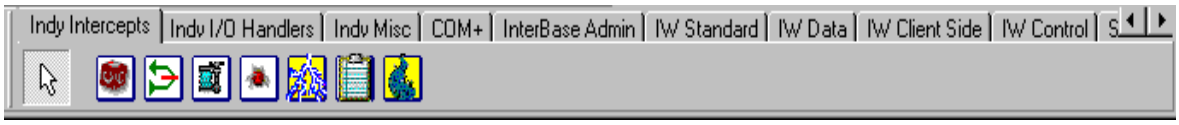
Indy Clients

Los componentes en la pestaña Indy Clients de la paleta de componentes permiten desarrollar aplicaciones cliente que utilicen los protocolos más populares de Internet.

Indy Servers

Los componentes en la pestaña Indy Servers permiten desarrollar aplicaciones servidoras que utilicen los protocolos más populares de Internet.

Indy Intercepts



Los componentes en la pestaña Indy Intercepts permiten desarrollar aplicaciones que utilicen los protocolos más populares de Internet.

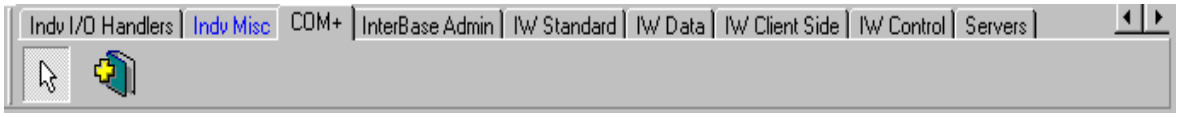
Indy I/O Handlers

Los componentes en la pestaña Indy I/O Handlers permiten desarrollar manejadores para aplicaciones que utilicen los protocolos más populares de Internet.

Indy Misc

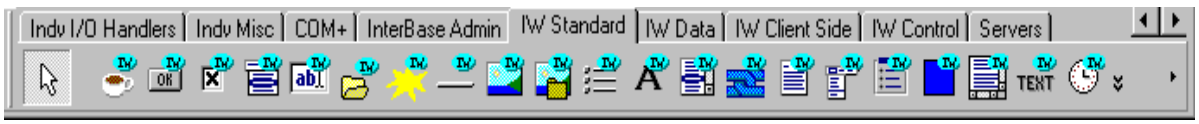
Los componentes en la pestaña Indy Misc permiten desarrollar aplicaciones que utilicen los protocolos más populares de Internet.

COM+



Los componentes en la pestaña COM+ incluye un único componente denominado TCOMAdminCatalog, el cual permite a una aplicación actuar como un controlador de Automation para la herramienta de administración de COM+.

IW Standard

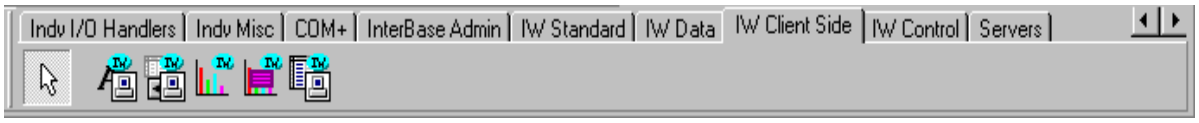


Los componentes en la pestaña IW Standard representan controles estándar en aplicaciones de IntraWeb.

IW Data

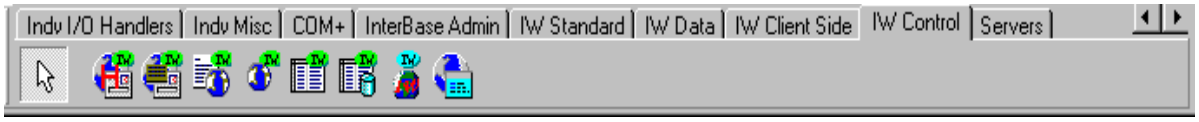
Los componentes en la pestaña IW Data representan controles de acceso a datos en aplicaciones de IntraWeb.

IW Client Side



Los componentes en la pestaña IW Client Side representan controles del lado del cliente en aplicaciones de IntraWeb.

IW Control



Los componentes en la pestaña IW Control representan componentes no visuales en aplicaciones de IntraWeb.

Servers

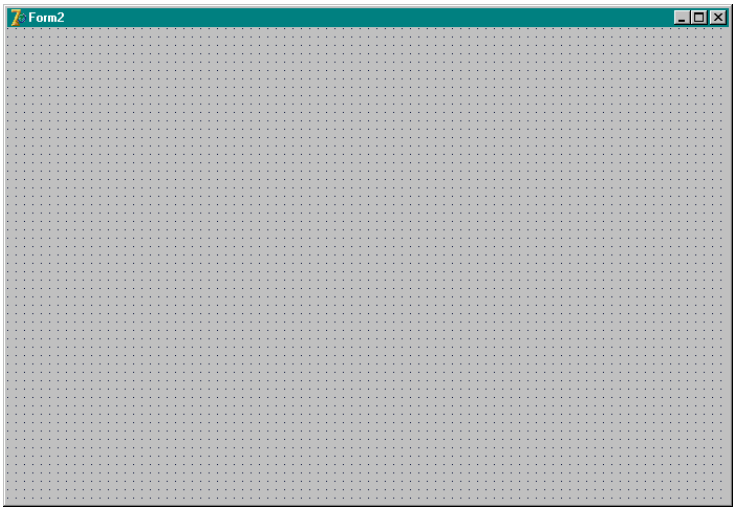
Los componentes en la pestaña Servers representan contenedores de VCL para servidores comunes de COM (Microsoft Office).

InterBase Admin

Los componentes en la pestaña InterBase Admin permiten invocar los servicios de APIs de Interbase.

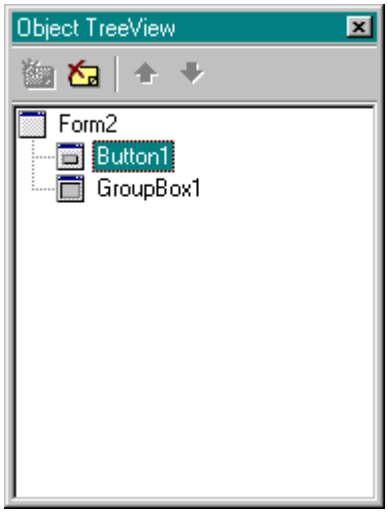
2.5 La Forma

La forma permite construir en tiempo de diseño las ventanas de una aplicación. Colocando componentes sobre una forma desde la paleta de componentes, se construye la interfaz gráfica de usuario de una aplicación. La siguiente figura muestra el aspecto general de una forma.



2.6 El Árbol de Objetos

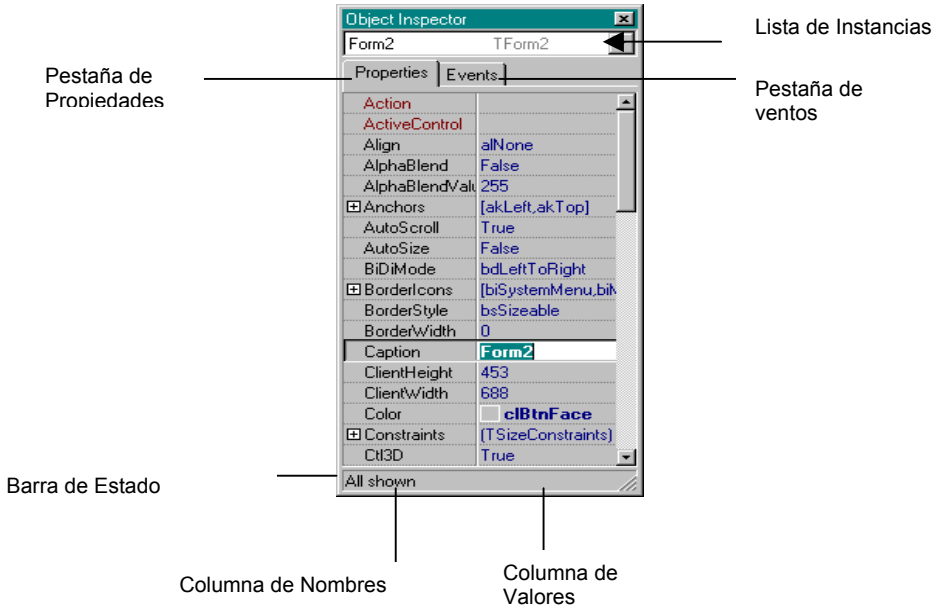
El Árbol de Objetos despliega un diagrama de los componentes visuales y no visuales que se encuentran en una forma, data module o frame, mostrando las relaciones lógicas entre componentes.



2.7 El Inspector de Objetos

El inspector de Objetos permite desarrollar una aplicación en tiempo de diseño, interactuando con las formas y los componentes que contienen, modificando sus propiedades y generando manejadores de eventos. Así mismo, el Inspector de Objetos facilita la selección de objetos no visibles dentro de la forma utilizando el Selector de Objetos.

El Inspector de Objetos se muestra en la siguiente figura:



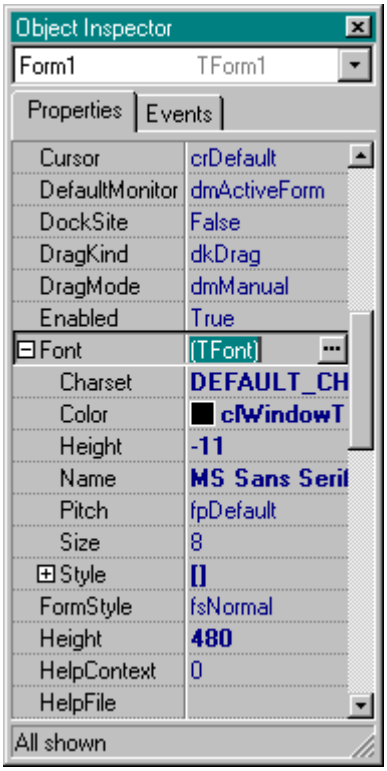
La ventana del Inspector de Objetos posee dos pestañas, la pestaña de propiedades: Properties y la pestaña de eventos: Events.

Todo componente posee un estado y un comportamiento, donde el estado está representado por medio de una lista de atributos o propiedades que posee el componente y el comportamiento está representado por una serie de métodos y eventos.

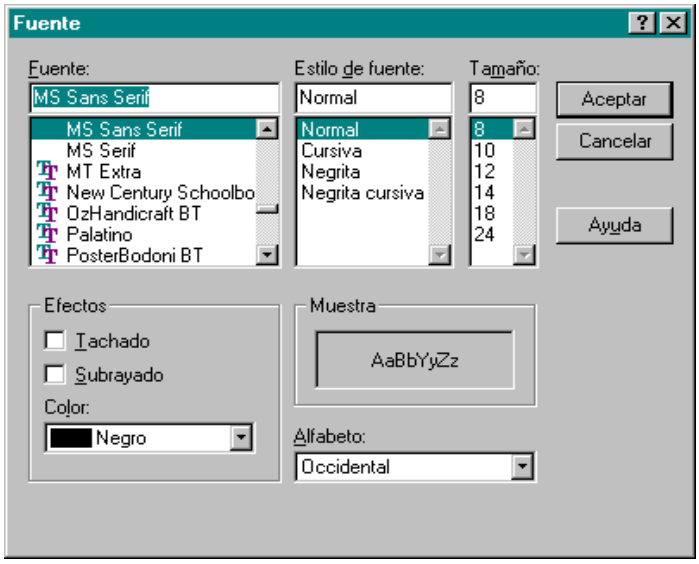
A través de la pestaña Properties podemos modificar el estado de un componente al modificar el valor de una o más de sus propiedades.

Dependiendo del tipo de propiedad seleccionada, el inspector de objetos mostrará un tipo de Editor de Propiedad adecuado, por ejemplo, si selecciona la propiedad Font del botón, en el extremo izquierdo de la propiedad Font se

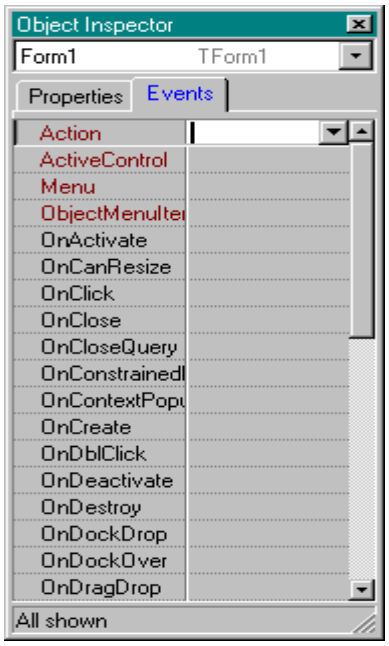
mostrará un signo de +, esto indica que la propiedad contiene un conjunto de subpropiedades. Se puede pulsar click en el signo de + para expandir la propiedad y tener acceso a dicho conjunto de subpropiedades.



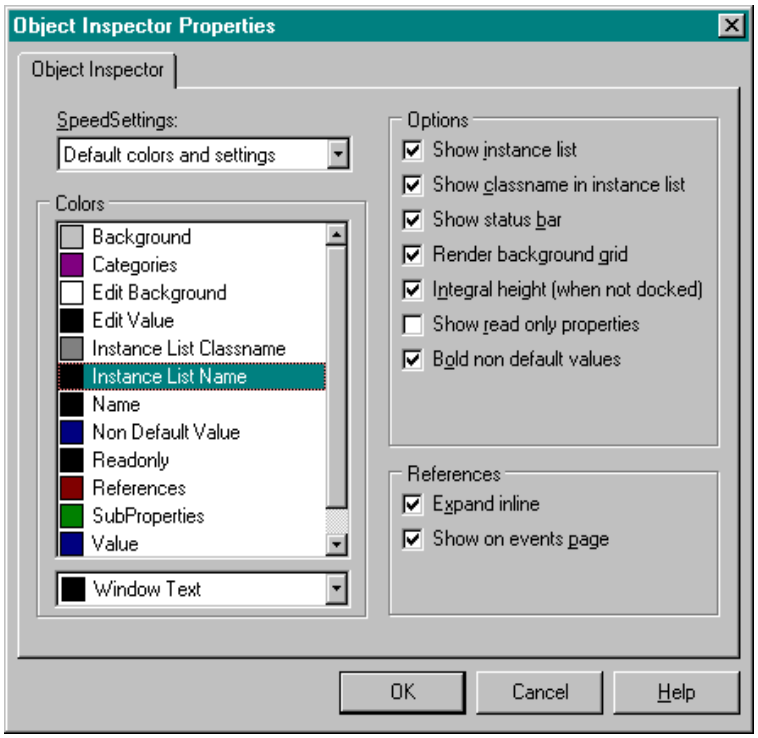
Existen propiedades que en el extremo derecho presentan un botón con tres puntos (…), como la propiedad Font, al pulsar click sobre éste botón se muestra un cuadro de diálogo el cual permite modificar el conjunto de valores de la propiedad.



Atraves de la pestaña Events del inspector de objetos podemos modificar el comportamiento de un componente generando manejadores de eventos.



El inspector de objetos puede personalizarse pulsando click derecho del Mouse sobre éste y seleccionando la opción properties, para mostrar el siguiente cuadro de diálogo.



En la sección SpeedSettings de este cuadro de diálogo se pueden configurar los colores con los que se muestran los distintos tipos de elementos del Inspector de Objetos.

La sección Options permite modificar los siguientes elementos:

Show instante list:

Permite mostrar u ocultar la Lista de Instancias.

Show classname in instant list:

Permite mostrar u ocultar el nombre de la clase del objeto seleccionado en la Lista de Instancias

Show status bar:

Permite mostrar u ocultar la barra de estado del Inspector de Objetos.

Render background grid:

Permite mostrar u ocultar líneas horizontales y verticales para delimitar las propiedades y eventos.

Integral height (when not docked):

Conforme se redimensiona el Inspector de Objetos con el cursor del Mouse, esta opción ajusta el inspector de Objetos entre una fila completa en lugar de una fila parcialmente mostrada.

Show read oniy properties:

Permite mostrar u ocultar propiedades de solo lectura.

Bold non default values:

Permite mostrar en negritas aquellas propiedades cuyos valores sean distintos de sus valores por omisión (default).

La sección References de éste cuadro de diálogo permite modificar los valores de las referencias a otros componentes mediante las siguientes opciones.

Expand in line:

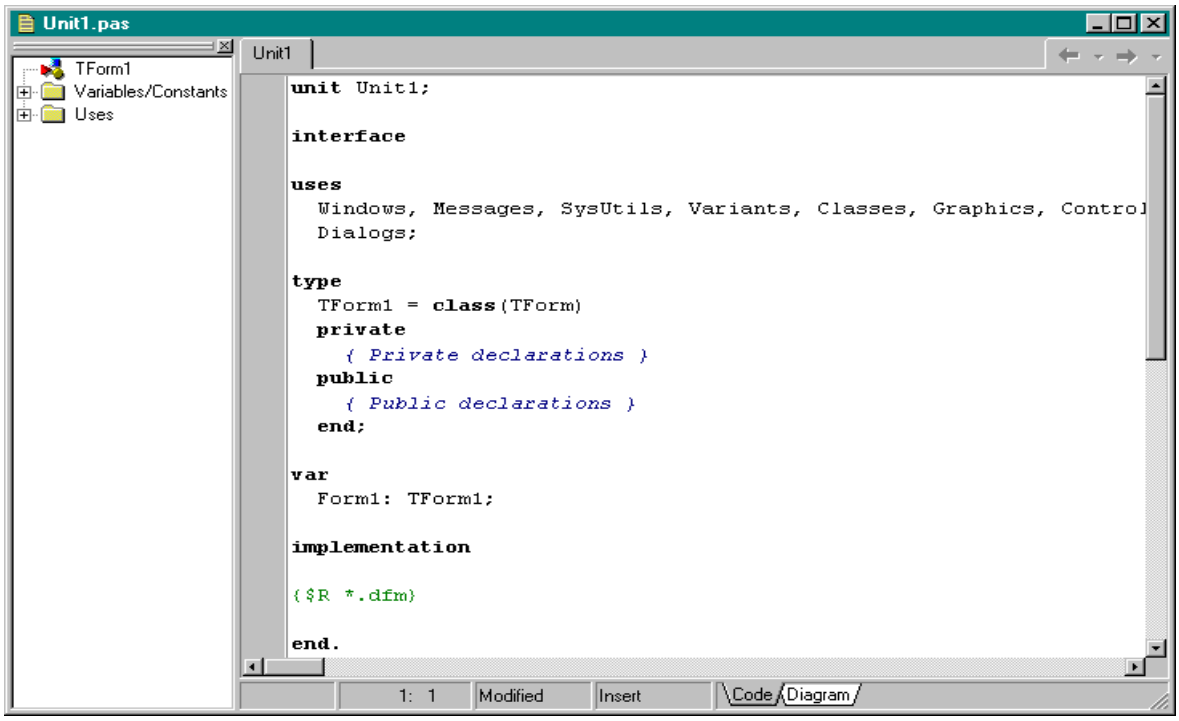
Despliega las propiedades del componente al cual se hace referencia.

Show on events page:

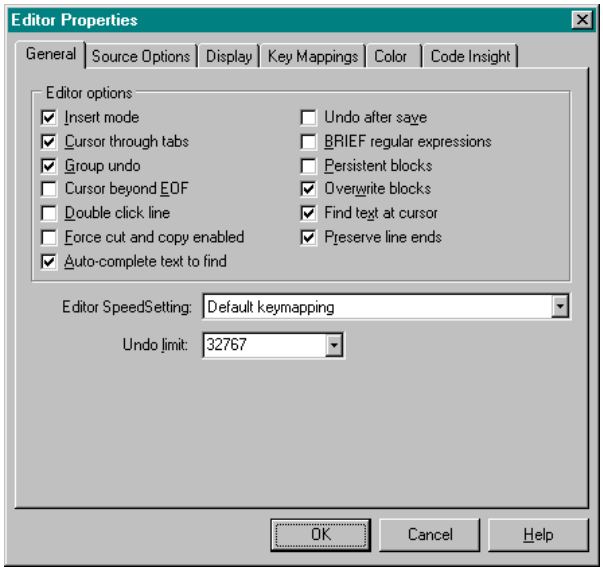
Despliega los eventos del componente al cual se hace referencia.

2.8 El Editor de Código

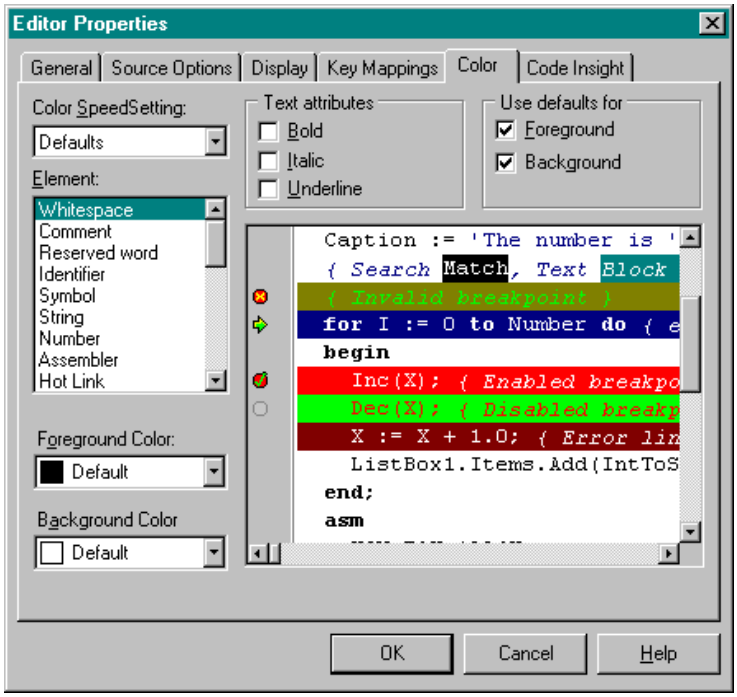
El Editor de Código es el lugar donde se introduce el código fuente de las aplicaciones.



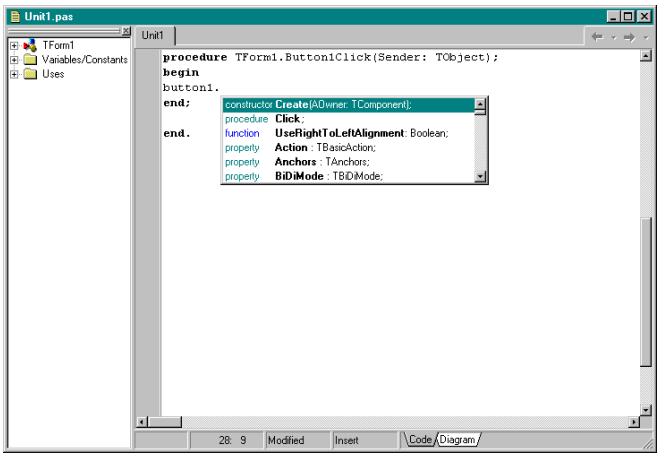
El editor de Código posee diversas características las cuales se pueden configurar a través del menú Tools | Editor Options.



Por medio de este cuadro de diálogo se pueden configurar entre otras opciones el realce de sintaxis de colores el cual en Delphi 7 ahora permite seleccionar colores personalizados para distintas instrucciones del lenguaje, como se muestra en la siguiente figura.

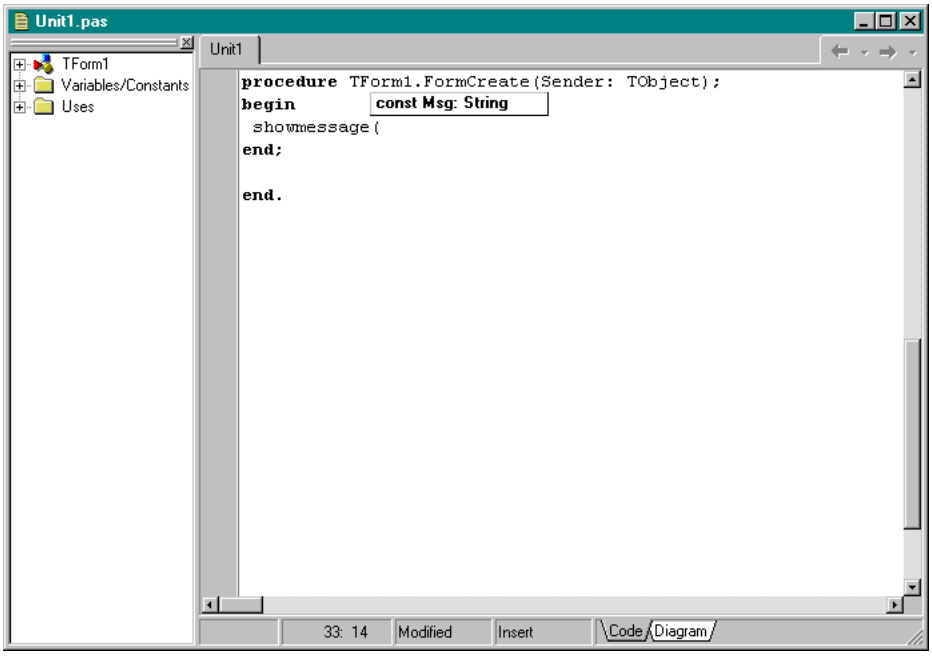


Una de las herramientas más útiles del Editor de Código es el Code Completion, el cual muestra una ventana que lista distintos elementos que forman parte de nuestra aplicación, así como sus procedimientos, funciones, variables, métodos, etc. Y a los cuales se hace referencia en el código fuente, por ejemplo, la siguiente figura muestra los distintos métodos y propiedades de la clase TButton.



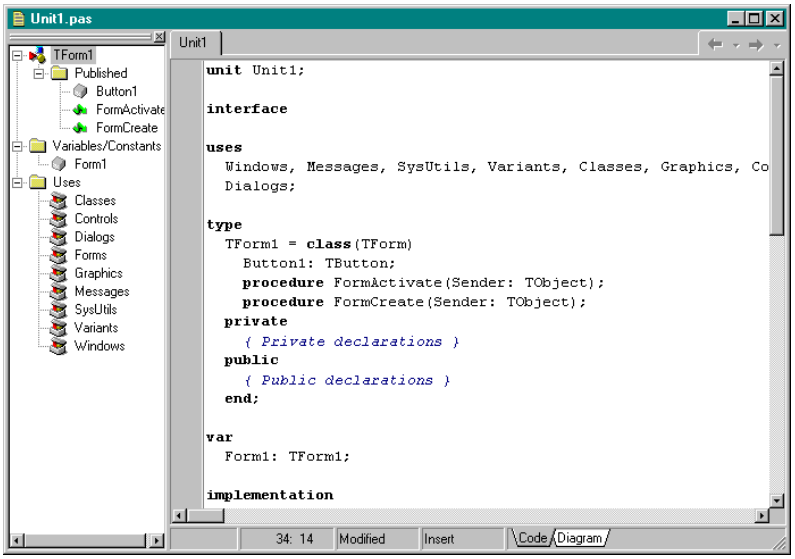
Las opciones mostradas en la ventana de code completion se presentan con distintos colores para diferenciar entre las funciones, procedimientos, propiedades, etc.

Además el code completion muestra los parámetros y sus tipos que espera un procedimiento o función, tal como se muestra en la siguiente pantalla.



2.9 "El explorador de Código

Esta herramienta se utiliza para navegar a través de las diferentes secciones que integran una unidad de código de un proyecto, por omisión, el explorador de código está incrustado a la izquierda dentro de la unidad de código.



El explorador de código contiene un diagrama de árbol, el cual muestra los tipos, clases, propiedades, métodos, variables globales, etc. definidas en una unidad, además muestra las unidades listadas en la sección uses. Al pulsar doble click sobre un elemento del árbol del explorador de código, el cursor se posiciona en el elemento seleccionado dentro de la unidad de código.

El Explorador de Código utiliza los siguientes elementos:

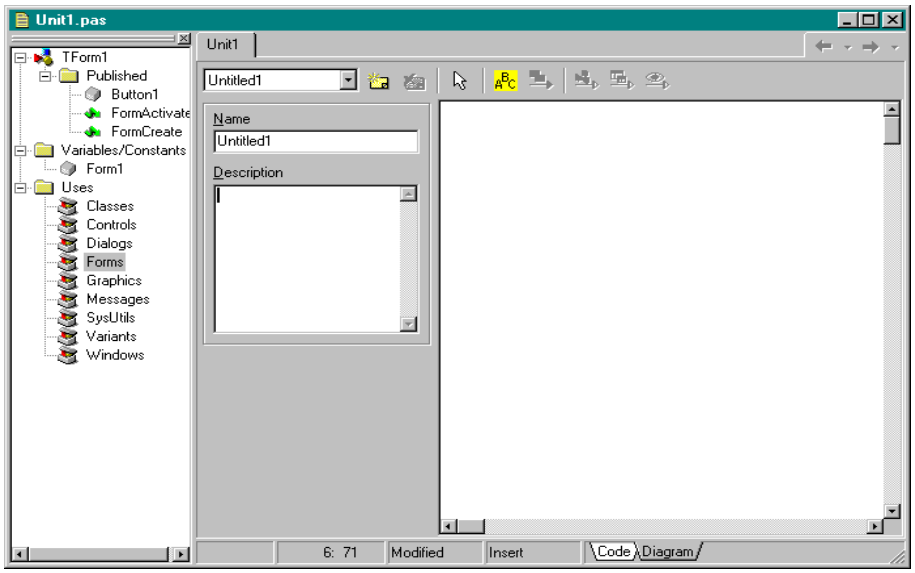
- **Clases**
- **Interfaces**
- **Unidades**
- **Constantes o variables (incluyendo campos)**
- **Métodos o rutinas (procedimientos en verde)**
- **Métodos o rutinas (funciones en amarillo)**
- **Propiedades**
- **Tipos"**[2]

[2] Ayuda del Software Delphi 7

2.10 La Ventana de Diagrama

La página Diagram, en el editor de código, provee una herramienta visual para configurar uno o varios diagramas que muestren la relación entre los componentes incluidos en la forma, además de permitir establecer comentarios sobre estas relaciones.

Para incorporar un componente dentro del diagrama, es necesario arrastrarlo desde la ventana del Object Tree View, pudiendo en su caso, seleccionar múltiples componentes desde esta ventana y agregarlos dentro de la página Diagram.



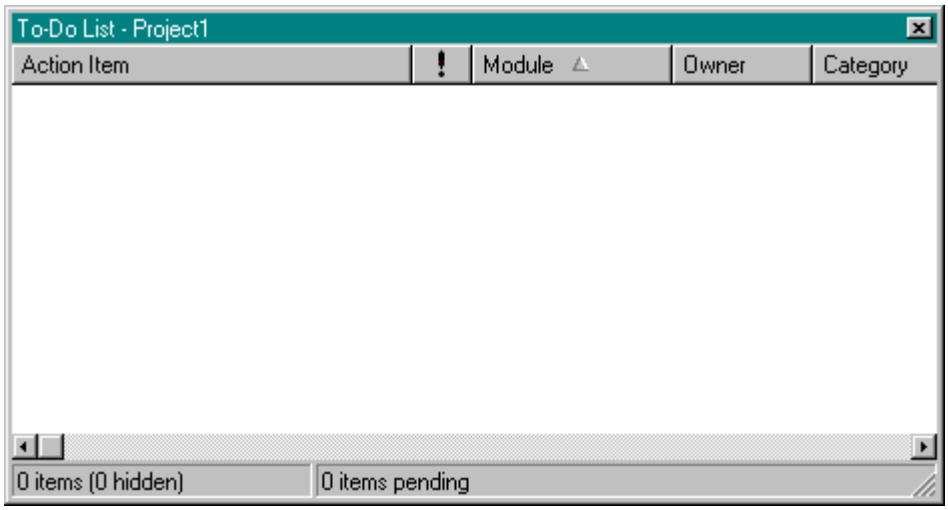
La parte izquierda de la página Diagram tiene una pequeña barra de herramientas que permite seleccionar, crear o eliminar el diagrama de trabajo actual. En la parte inferior de dicha barra se puede asignar un nombre a cada nuevo diagrama y asociarle una descripción.

La barra de botones para generar las relaciones y comentarios se encuentra en la parte superior de la página de diagrama.

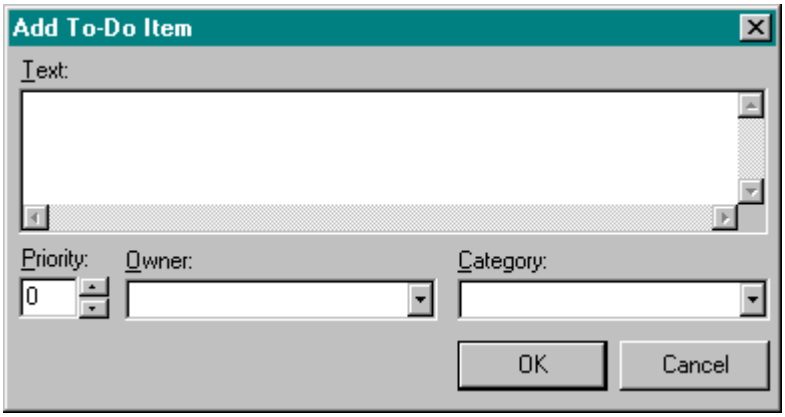
2.11 Herramientas de Apoyo

La Lista de Pendientes

La lista de pendientes (To-do list) permite almacenar notas de pendientes que necesitan resolverse en un proyecto, se pueden agregar pendientes directamente a la lista, o se pueden agregar pendientes directamente al código fuente; la lista de pendientes se muestra en la siguiente figura.



Para agregar un pendiente a la lista, se debe pulsar click derecho del Mouse sobre la ventana de la lista de pendientes y seleccionar la opción Add del menú contextual.



En este cuadro de diálogo se debe introducir la siguiente información:

Text.- Una descripción del pendiente.

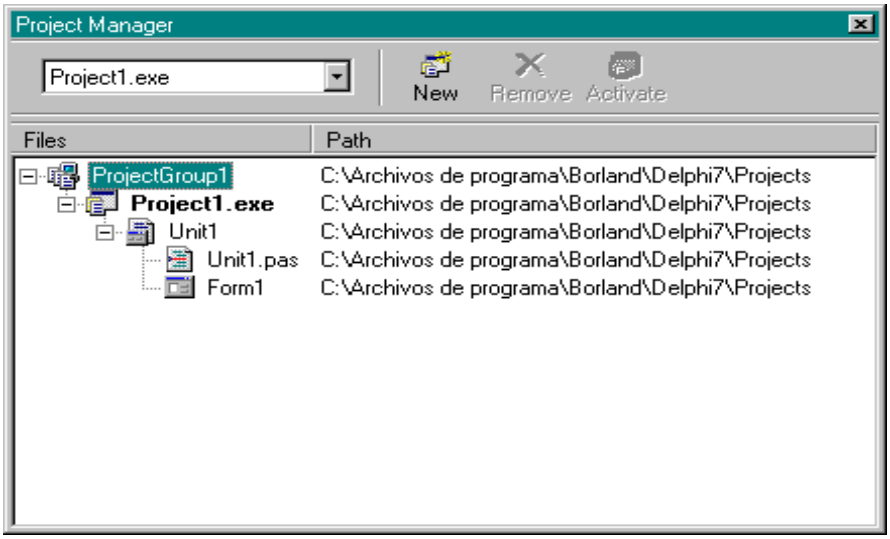
Priority.- Nivel de prioridad del pendiente.

Owner.- Responsable de completar el pendiente.

Category.- Indica el tipo de pendiente.

El Manejador de Proyectos

El manejador de proyectos despliega información acerca del estado y los archivos de los que consta el proyecto actual, si el proyecto es parte de un grupo de proyectos, despliega información acerca de todos los proyectos de dicho grupo.



Por medio del manejador de proyectos se puede visualizar como están relacionados un grupo de proyectos, se puede seleccionar cualquier archivo de un proyecto, pulsar click derecho sobre él y ejecutar una serie de acciones como abrir el archivo o removerlo del proyecto; También es posible agregar archivos existentes a un proyecto.

2.12 Preguntas y Ejercicio de Repaso

1. ¿Qué es la Paleta de Componentes?

2. Encuentra la definición que corresponda a cada evento.

Use Unit

Permite compilar los módulos de compilación del proyecto actual

Bring to Front

Muestra la ventana del navegador de clases.

Tab Order

Permite utilizar una unidad lógica de código desde otra unidad.

Find Error

Coloca un control al frente de otros controles.

Browser Shift+Ctrl+B

Permite buscar un error en el proyecto actual.

Syntax check [Proyecto]

Permite modificar el orden de tabulación de los controles de un contenedor.

3. Mencione la función del Inspector de Objetos.

4. ¿Cuales son las barras de herramientas con las que cuenta Delphi?

5. ¿Cuál es la Finalidad del Arbol de Objetos?

6. ¿Qué es el Explorador de Código?

7. Mencione la función de la lista de pendientes.

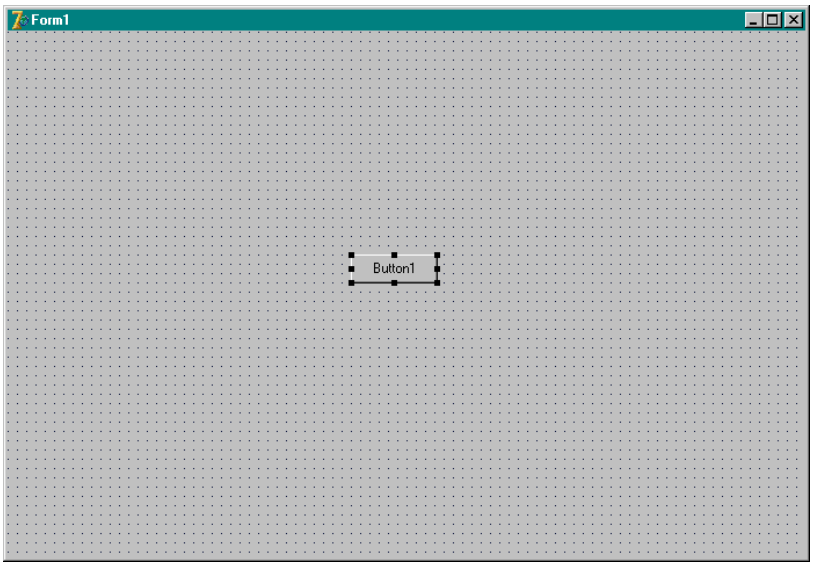
8. Seleccione dos elementos que utiliza el Explorador de Codigo

a)Clases y Propiedades b) Unidades y Standard c)Clases y Rave

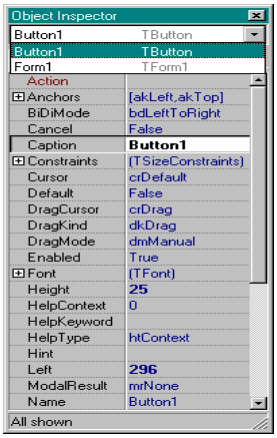
Ejercicio de Repaso

A continuación desarrollaremos una aplicación para mostrar la manera de trabajar con el Inspector de Objetos y sus características.

1. Seleccione el menú File | New | Application. Delphi generará una aplicación nueva automáticamente.
2. Agregue un botón a la única forma de nuestra aplicación, para esto, seleccione la pestaña Standard de la paleta de componentes, localice el componente Button, pulse click sobre él y después pulse click sobre la forma. Su forma deberá tener el siguiente aspecto:



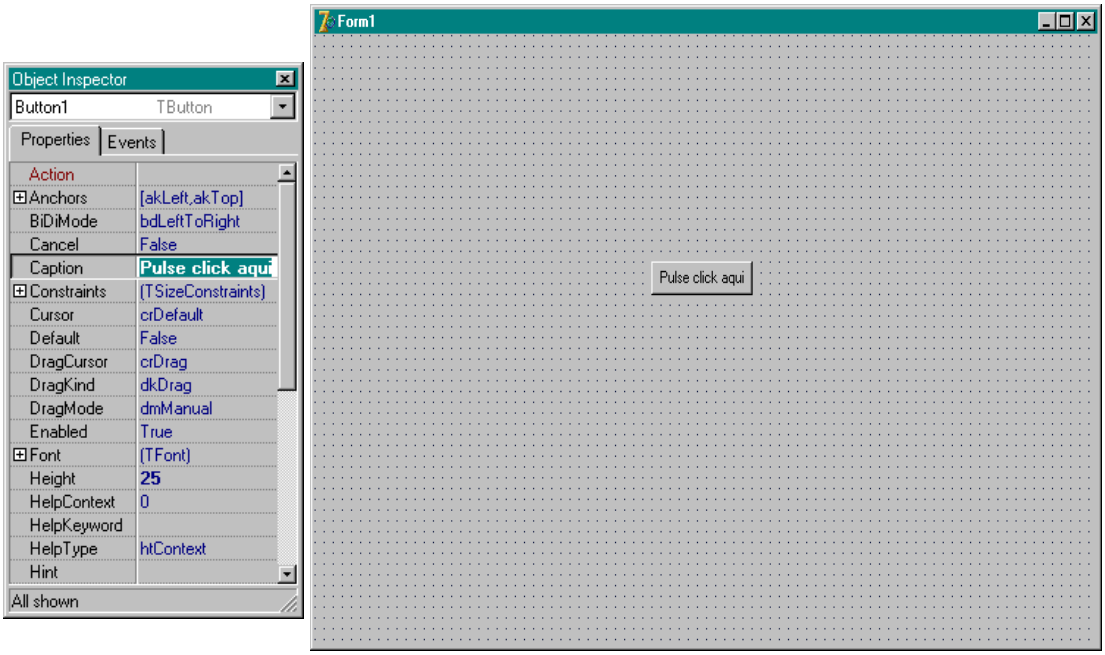
3. En la parte superior del inspector de objetos, se encuentra la Lista de Instancias, la cual permite seleccionar entre una forma y los componentes contenidos en la misma, seleccionar entre una forma y los componentes contenidos en la misma, seleccione de esta lista el valor Button1, tal como se muestra en la siguiente pantalla.



Con esta acción le indicamos el inspector de objetos que deseamos seleccionar el botón que previamente agregamos a la forma para modificar sus propiedades y eventos.

Al seleccionar el botón dentro de la lista de instancias, el inspector de objetos muestra las propiedades definidas para el componente Button1, el cual es una instancia de la clase Tbutton, los conceptos de clase e instancia se definirán más adelante.

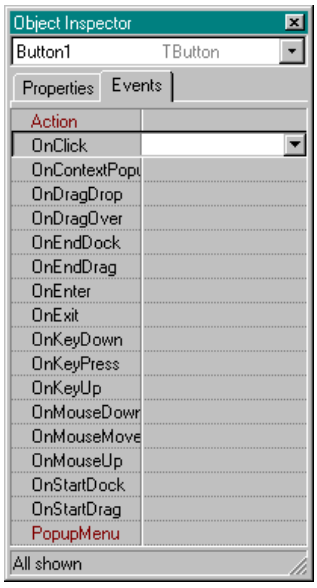
4. Seleccione la propiedad Caption del botón pulsando click sobre ésta, y modifique su valor por: Pulse click aquí, note que al modificar el valor de la propiedad también se modifica la representación visual del botón, tal como se muestra en la siguiente figura.



5. Modifique la propiedad Font del botón tal como se muestra a continuación.

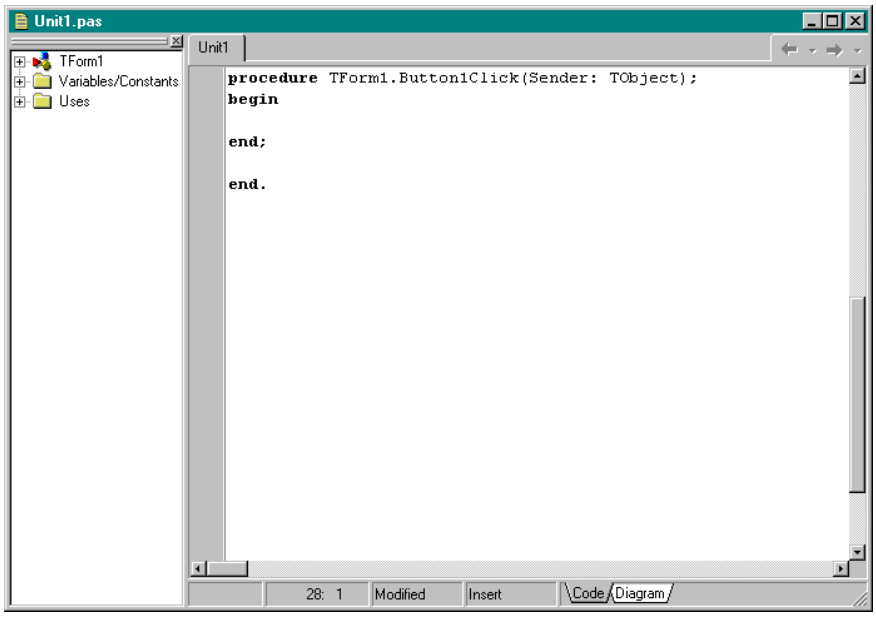
Note que el texto del botón es demasiado grande para el tamaño del botón, por lo que debemos modificarlo.

6. Seleccione la propiedad Width del botón y modifíquela a un valor de 150.
7. Seleccione la propiedad Height del botón y modifíquela a una valor de 35.
8. Seleccione la pestaña Events, ésta muestra la lista de eventos definidos para el botón.



9. Seleccione el evento OnClick del botón y pulse doble click sobre la sección en blanco que se muestra a la derecha de éste, Delphi generará automáticamente un manejador de evento en la Unidad de Código, como se muestra a continuación.

   La unidad de código la analizaremos más adelante.



10. Escriba la siguiente línea de código en el lugar donde se ubica su cursor, dentro de la unidad de código:

    Form1.Caption := 'Mi primera aplicación en Delphi';

Con este código estamos indicando que al ejecutar nuestra aplicación y pulsar click en el botón de Caption de la forma se modificará por: Mi primera aplicación en Delphi.

11. Ejecute su aplicación presionando la tecla F9

12. Pulse click sobre el botón, el texto en la barra de título de la ventana de la aplicación deberá modificarse.

13. Cierre su aplicación para regresar al Ambiente de Desarrollo de Delphi.

III. El Lenguaje Object Pascal

3.1 Introducción

"El lenguaje Object Pascal, utilizado por Delphi, es una extensión del lenguaje Pascal creado por Niklaus Wirth en 1971, Pascal fue diseñado como una versión simplificada del lenguaje Algol (el cual se diseñó en 1960) para propósitos educacionales en clases de programación.

En 1983 Borland desarrolló su compilador Turbo Pascal, que ha sido uno de los más vendidos de todos los tiempos, lo cual contribuyo a que el lenguaje se volviera popular en las PC's. Después de nueve versiones del compilador, las cuales gradualmente agregaron extensiones al lenguaje, en 1995 Borland liberó Delphi, que convirtió a Pascal en un lenguaje de programación visual.

Object Pascal agrega soporte de programación orientada a objetos (POO), los conceptos esenciales de la programación orientada a objetos."[1]

A continuación se describirán las instrucciones básicas del lenguaje Object Pascal.

3.2 Comentarios

Todo lenguaje de programación debe soportar la generación de comentarios en el código fuente de un programa y Object Pascal no es la excepción, los comentarios tienen como propósito describir el código fuente de un programa de tal manera que este sea más sencillo de comprender, facilitando su mantenimiento.

Cuando se compila un programa, el compilador ignora el código marcado como comentarios, de tal manera que este no es compilado.

Object Pascal soporta distintos tipos de comentarios, los cuales se describen a continuación:

Comentarios de una línea

Se puede comentar el contenido de una línea de código por medio de los caracteres "//", todo lo que aparezca a la derecha de dichos caracteres será tratado como un comentario, tal como se muestra en el siguiente ejemplo.

```
// Este mensaje es un comentario de una línea de código
```

[1] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

“Comentarios de varias líneas de código

Un comentario puede abarcar más de una línea de código, para definirlos se pueden utilizar los delimitadores "{" y"}". o bien "(*" y "*)", como se muestra a continuación:

{ Este es un comentario
de varias líneas de
extensión }

(* Este es otro comentario.
de varias líneas de
extensión *)”[2]

3.3 Variables

Una variable es un identificador cuyo valor puede modificarse en tiempo de ejecución, técnicamente una variable es un nombre asociado a una localidad de memoria, se puede utilizar dicho nombre para leer o escribir el valor de la localidad de memoria, las variables son contenedores de datos, y poseen un tipo, de tal forma que le pueden indicar al compilador como interpretar el dato que contienen.

La sintaxis para declarar variables es la siguiente:
Var Identificador: tipo;

Por ejemplo, la siguiente línea de código:
Var num: Integer;
Declara una variable de tipo Integer llamada num.

También es posible declarar un grupo de variables de un mismo tipo en una única línea de código, para esto se emplea la siguiente sintaxis:
Var ListaIdentificadores; tipo;

Donde ListaIdentificadores es una lista separada por comas de identificadores válidos y tipo es un tipo de dato válido, por ejemplo, la siguiente línea de código:
Var x, y: Double;

Declara dos variables de tipo Double llamadas x y y respectivamente.

[2] http://www.marcocantu.com/EPascal/Spanish/ch02code.htm

“Variables locales y Variables globales

Las variables declaradas dentro de una función o procedimiento se denotan como variables locales, mientras que las variables que no se declaran dentro de un procedimiento o función reciben el nombre de variables globales. Las variables globales pueden inicializarse al mismo tiempo que se declaran utilizando la sintaxis:

Var Indentificador : tipo = Expresion Constante;

Donde ExpresionConstante es una expresión constante representando un valor del tipo de la variable, por ejemplo, la siguiente línea:

Var num: Integer = 15;

es equivalente a la declaración y sentencia:

Var num: Integer;

...

num : = 15;

Si no sé inicializa explícitamente una variable global, el compilador le asigna automáticamente un valor de 0 (o nil sí es un objeto), mientras que las variables locales no pueden inicializarse en su declaración y contienen valores aleatorios hasta que se les asigne un valor.”[3]

3.4 Constantes

Distintas construcciones del lenguaje reciben el nombre de constantes, por ejemplo existen constantes numéricas como 3, y constantes de cadena como "Hola Mundo". Cada tipo enumerado define constantes que representan los valores de dicho tipo; existen constantes predefinidas como True, False o nil. Finalmente existen constantes, que al igual de las variables, se crean individualmente por declaración.

Las constantes declaradas pueden ser constantes verdaderas o constantes de tipo, aunque ambos tipos parecen similares, están gobernadas por reglas diferentes y se utilizan para propósitos distintos.

Constantes Verdaderas

Una constante verdadera está definida como un identificador cuyo valor no puede ser modificado, la sintaxis para declarar una constante verdadera es;

const identificador = ExpresionConstante;

[3] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

Donde identificador es un identificador válido y ExpresionConstante es una expresión que el compilador puede evaluar sin ejecutar el programa.

Por ejemplo:
const NumItems = 134;

Declara una constante llamada Numltems que devuelve el valor 134.

Si ExpresionConstante devuelve un valor ordinal, se puede especificar el tipo de la constante forzando el tipo. por ejemplo:
const num = Int64(15);

Declara una constante llamada num, de tipo lnt64. que devuelve el valor 15, de otra forma el valor de la constante sería el tipo devuelto por ExpresionConstante, por ejemplo:
const saludo = 'hola';

Declara una constante llamada saludo, que devuelve el valor "Hola", de tipo cadena.

Sí ExpresionConstante es una cadena de caracteres, la constante declarada es compatible con cualquier tipo de cadena, si la cadena de caracteres está formada por un único carácter, la constante también es compatible con el tipo char.

Si ExpresionConstante es un número de coma flotante su tipo es Extended, mientras que si es un entero, su tipo está dado por la siguiente tabla.

RANGO DE CONSTANTE (HEXADECIMAL)	RANGO DE CONSTANTE (DECIMAL)	TIPO
-$8000000000000000..-$80000001	-2^63..-2147483649	lnt64
-$80000000..-$8001	-2147483648..-32769	Integer
-$8000..-$81	-32768..-129	Smallint
-$80..-1	-128..-1	Shortint
0..$FF	0..255	Byte
$100..$FFFF	256.. 65535	Word
$10000..$FFFFFFFF	65536 . 4294967295	Cardinal
$100000000.,$7FFFFFFFFFFFFFFF	4294967296..2^63-1	lnf64

El siguiente ejemplo muestra diferentes declaraciones de constantes.

```
Const
  izq   =  0;
  der = 100;
  Centro = (izq. - der.) div 2;
  Letra – Chr (225);
  Error = 'Memoria agotada';
  ErrStr = ' Error: ' +  Message + '  ';
  Ln10 = 2.302585052994045684;
  Ln10R = 1 / Ln10;
  Num = [ ' 0 '. . ' 9'];
```

Constantes de Tipo

Las constantes de tipo pueden almacenar tipos de arreglos, registros, procedimientos y apuntadores, las constantes de tipo no pueden ocurrir en expresiones constantes, la sintaxis para declarar una constante de tipo es:
Const identificador: Tipo = valor;

Donde identificador es un identificador válido, tipo es cualquier tipo con la excepción de archivos y variants y valor es una expresión valida del tipo definido.
Por ejemplo:
Const num: Integer = 300;

Casi siempre la expresión valor, debe ser una expresión constante, sin embargo sí tipo es un arreglo, registro, procedimiento, o apuntador se utilizan reglas especiales.

3.5 Operadores

"Los operadores se comportan como funciones predefinidas, por ejemplo la expresión (A+B) está formada por las variables A y B, con el operador +, si en este ejemplo, A y B representan enteros o números en coma flotante (A+B) devuelve su suma.

Ejemplos de operadores son:
@, not, ^, *, /, div, mod, and, shl, shr, as, is, +, -, or, xor, =, >, <, <>, <=. >=, in."[4]

[4] Ayuda del Software Delphi 7

Contrarío a la mayoría de los lenguajes de programación, los operadores and y or tienen precedencia comparada a la de los operadores relaciónales, por ejemplo si escribimos la expresión a<b and c<d, el compilador tratará de ejecutar la operación and primero, resultando en un error de compilación, por lo que se debe de escribir la expresión en la forma (a<b) and (c<d).

Algunos operadores tienen distintos comportamientos con diferentes tipos de datos. Por ejemplo, el operador + se puede utilizar para sumar dos números, concatenar dos cadenas, hacer la unión de dos conjuntos y agregar un desplazamiento a un apuntador PChar.

Otro operador especial en Pascal es el operador div, por ejemplo, se pueden dividir dos números cualesquiera con el operador /, e invariablemente el resultado será un número real, si se necesita dividir dos números enteros y se desea que el resultado sea un número entero se debe utilizar el operador div en lugar de /.

La siguiente tabla muestra los operadores de Object Pascal, agrupados por precedencia.

OPERADORES UNARIOS (PRECEDENCIA MÁS ALTA)	
@	Dirección de la variable o función (devuelve un apuntador).
Not	Negación booleana o de bits.
OPERADORES MULTIPLICATIVOS Y DE BITS	
*	Multiplicación aritmética o intersección de conjuntos.
/	División de punto flotante.
Div	División de enteros.
Mod	Módulo (residuo de la división de enteros).
As	Permite conversión de tipos verificada en tiempo de ejecución.
And	And booleano o de bits.
Shl	Corrimiento de bits a la izquierda.
Shr	Corrimiento de bits a la derecha.
OPERADORES ADITIVOS	
+	Adición aritmética, unión de conjuntos, concatenación de cadenas, Adición de desplazamiento de apuntadores.
-	Substracción aritmética, diferencia de conjuntos, substracción de Desplazamiento de apuntadores.
Or	Or booleano o de bits.
Xor	Or exclusivo booleano o de bits

OPERADORES RELACIONALES Y DE COMPARACION (PRECEDENCIA MÁS BAJA)	
=	Operador de igualdad.
<>	Operador de desigualdad.
<	Operador menor que.
>	Operador mayor que.
<=	Operador menor o igual que, subconjunto de un conjunto.
>=	Operador mayor o igual que, superconjunto de un conjunto.
In	Verifica si un elemento es miembro de un conjunto.
Is	Verifica si el tipo de un objeto es compatible a otro.

3.6 Procedimientos y Funciones

Un concepto importante en Pascal es el de rutina, una rutina está formada por una serie de sentencias bajo un nombre único, la cual puede ser activada muchas veces utilizando su nombre, evitando duplicar las mismas sentencias una y otra vez, y facilitando la modificación del programa. Las rutinas son una forma básica de encapsulamiento.

Existen dos tipos de rutinas en Object Pascal: los procedimientos y las funciones. Técnicamente un procedimiento es una operación que se le pide a la computadora que efectué, mientras que una función es un cálculo que devuelve un resultado, la principal diferencia es que un procedimiento no devuelve un valor.

Sin embargo la diferencia entre procedimientos y funciones es limitada, por ejemplo el llamado a una función puede ignorar el resultado devuelto por la función, o se puede declarar un procedimiento que pase un resultado dentro de sus parámetros.

A continuación se muestra una declaración de procedimiento y dos formas distintas de declarar una función.

```
Procedure HolaMundo;
begin
   ShowMesssage ( 'Hola Hundo');
end;

function Triple (Value: Integer) : Integer;
begin
```

```
    Triple : = Value * 3;
  end;
```

O bien:

```
  function Triple (Value : Integer) : Integer;
  begin
    Result := Value * 3;
  end;
```

El uso de Result en lugar del nombre de la función para designar el valor de retorno enfatiza que la función devuelve un valor.

Una vez que se han declarado estas rutinas pueden llamarse una o tantas veces como sea necesario.
Para probar los conocimientos adquiridos realizaremos el siguiente ejercicio:

Aplicación de Ejemplo.

1. Cree una nueva aplicación en Delphi por medio del menú File | New | Application.
2. Agregue dos componentes Button y un componente Edit de la pestaña Standard en la forma de su aplicación
3. En la sección Implementation de su unidad de código implemente los siguientes procedimientos:

```
implementation
{$R *.dfm}
Procedure HolaMundo;
begin
     ShowMessage('Hola Mundo') ;
end;

function Triple(Value: Integer): Integer;
begin
   Result := Value * 3;
end;
```

4. En el evento OnClick del componente Button1, implemente el siguiente código, a través del Inspector de Objetos, pulsando doble click sobre el valor del evento OnClick:

```
procedure Tform1.Button1Click(Sender: TObiect) ;
begin
   HolaMundo;
```

```
    end;
```

5. Y por último, en el evento OnClick del componente Button2, implemente el siguiente código:

```
    procedure Tform1.Button2Click(Sender: Tobject);
    var
    X, Y: Integer;
    begin
       X := Triple (StrToInt (Edit1.Text));
       Y := Triple (X) ;
      ShowMessage (IntToStr (Y));
   end;
```

6. Salve los cambios de la aplicación, su unidad de código deberá tener el siguiente aspecto:

```
unit Unit1;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls;

type
 TForm1 = class(TForm)
  Button1: TButton;
  Button2: TButton;
  Edit1: TEdit;
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);

 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Form1: TForm1;

implementation
{$R *.dfm}
```

```
procedure HolaMundo;
begin
ShowMessage('Hola Mundo') ;
end;

function Triple(Value: Integer): Integer;
begin
Result := Value * 3;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  HOLAMUNDO;
end;
procedure TForm1.Button2Click(Sender: TObject);
var
X, Y: Integer;
begin
X := Triple (StrToInt (Edit1.Text));
Y := Triple (X) ;
ShowMessage (IntToStr (Y));
end;
end.
```

7. Ejecute la aplicación presionando la tecla F9, y compruebe su funcionamiento.

3.7 Tipos de Datos

"Existen varios tipos de datos predefinidos en Pascal, los cuales se pueden dividir en tres grupos: tipos ordinales, tipos reales y cadenas. Delphi también incluye un tipo llamado variant el cual es un tipo de dato que no puede ser determinado en tiempo de compilación."[5]

Tipos Ordinales

Los tipos ordinales están basados en el concepto de orden o secuencia. No solamente nos permiten comparar dos valores para saber cual es el más grande, sino que podemos solicitar el valor anterior o siguiente de un valor dado, o calcular el mínimo o máximo valor posible.

[5] IDEM

Los tres tipos ordinales predefinidos más importantes son: Integer, Boolean y Char. Sin embargo existe un número de tipos relacionados que tienen el mismo significado, pero una representación interna distinta, así como un distinto rango de valores.

Tipos Enteros

La siguiente tabla, muestra los tipos de datos ordinales utilizados para representar números enteros.

TAMAÑO	RANGO CON SIGNO	RANGO SIN SIGNO
8 bits	Shortint -128...127	Byte O...255
16 bits	Smallint -32768...32767	Word O...65535
32 bits	Longint -2147483648... 2147483647	LongWord O...4294967295
64 bits	lnt64 -2^63...2^'63-1	
32 bits	Integer -2147483648...2147483647	Cardinal O...4294967295

Estos tipos corresponden a distintas representaciones de números, dependiendo del número de bits utilizados para representar el valor y si se encuentra presente o no un bit de signo. Los números con signo pueden ser positivos o negativos, pero poseen un rango menor de valores, porque disponen de un bit menos para el valor.

Los tipos Integer y Cardinal se utilizan frecuentemente, porque corresponden a la representación nativa de números en la CPU.

Tipos Booleanos

Los cuatro tipos booleanos predefinidos son: Boolean, ByteBool, WordBool y LongBool. Siendo Boolean el tipo preferido. Los otros tipos existen para proporcionar compatibilidad con otros lenguajes y librerías de sistemas operativos.

La siguiente tabla muestra el tamaño en bytes de los tipos booleanos.

TIPO	TAMAÑO
Boolean	1 byte
ByteBool	1 byte
WordBool	2 bytes
LongBool	4 bytes

Caracteres

Existen dos tipos diferentes de representación para los caracteres: ANSIChar y WideChar.

Las variables de tipo AnsiChar son caracteres de un byte (8 bits) de tamaño, ordenados de acuerdo al conjunto de caracteres local el cual es posiblemente de múltiples bytes. AnsiChar se modeló originalmente de acuerdo al conjunto de caracteres ANSI (de ahí su nombre), pero ahora ha sido expandido para referirse al conjunto de caracteres local.

Los caracteres WideChar son de dos bytes (16 bits) y están ordenados de acuerdo al conjunto de caracteres Unicode. Los primeros 256 caracteres Unicode corresponden con los caracteres ANSI.

Rutinas de Tipos Ordinales

La tabla siguiente muestra algunas rutinas de tipos ordinales.

RUTINA	PROPÓSITO
Dec	Decrementa la variable pasada como parámetro en uno, o en el valor de un segundo parámetro opcional.
Inc	Incrementa la variable pasada como parámetro en uno, o el valor de un segundo parámetro opcional.
Odd	Devuelve True si su argumento es un número impar.
Pred	Devuelve el valor anterior a su argumento en el orden determinado por su tipo ordinal.
Succ	Devuelve el siguiente valor a su argumento en el orden determinado por su tipo ordinal.
Ord	Devuelve un número indicando el orden de su argumento dentro del conjunto de valores de su tipo de dato.
Low	Devuelve el menor valor en el rango del tipo ordinal pasado como parámetro.
High	Devuelve el mayor valor en el rango del tipo ordinal pasado como parámetro.

3.8 Tipos Reales

Los tipos reales representan números en punto flotante en varios formatos. A Continuación se muestran los rangos de los distintos tipos reales.

TIPO	RANGO	DIG. SIGNIFICATIVOS	TAMAÑO
Real48	2.9x10^-39...1.7x10^38	11-12	6 bytes
Single	1.5x10^-45...3.4x10^38	7-8	4 bytes
Double	5.0x10^-324...1.7x10^308	15-16	8 bytes
Extended	3.6x10^-4951...1.1 x10^4932	19-20	10 bytes
Comp	-2^63+1...2^63-1	19-20	8 bytes
Currency	-922337203685477.5808... 922337203685477.5807	19-20	8 bytes

El tipo genérico Real, en su implementación actual, es equivalente a Double.

El tipo de 6 bytes Real48 se llamaba Real en versiones anteriores de Object Pascal. Si se necesita recopilar código que utilice el viejo formato de Real, se debe modificar a Real48, o bien se puede utilizar la directiva de compilación {$REALCOMPATIBILITY ON}, para interpretar el tipo Real en su viejo formato. El tipo Real48 se mantiene por razones de compatibilidad, se debería evitar su uso en aplicaciones nuevas porque su formato no es nativo en los procesadores de Intel, por lo que su uso resulta en un desempeño menor que los demás tipos reales.

Los tipos Single (4 bytes), Double (8 bytes) y Extended (10 bytes) corresponden al estándar IEEE de representación de números en punto flotante, y son soportados directamente por el coprocesador matemático de la CPU para un máximo desempeño.

El tipo Comp es nativo a los procesadores Intel y representa un entero de 64 bits. Sin embargo se clasifica como un tipo real porque su comportamiento no es el de un tipo ordinal (no se puede incrementar o decrementar un valor de tipo Comp). Este tipo se mantiene por razones de compatibilidad, en aplicaciones nuevas se debería de utilizar el tipo lnt64 para un mejor desempeño.

“Forzado y Conversión de Tipos

Generalmente no se puede asignar una variable a otra de un tipo distinto, sin embargo, si se necesita realizar esta acción se pueden emplear dos opciones. La primera opción es un forzado de tipos, la cual utiliza una notación funcional, con el nombre del tipo de dato de destino, el siguiente ejemplo ilustra este concepto:

```
var
  I: Integer;
  Ch: Char;
  found: Boolean;
```

```
begin
   I := Integer('X');
   Ch := Char(I);
   Found := Boolean(0);
end;
```

Se puede realizar un forzado de tipos entre tipos de datos que tengan el mismo tamaño. Normalmente es más seguro hacer un forzado de datos entre tipos ordinales, o entre tipos reales, también es posible forzar tipos de apuntadores y objetos.

Sin embargo el forzado de tipos se considera una práctica peligrosa, debido a que permite acceder un valor como si se estuviera representado de una forma distinta y debido a que las representaciones internas de los tipos de datos generalmente no coinciden, se pueden introducir errores difíciles de rastrear."[6]
La segunda opción es utilizar una rutina de conversión como las mostradas a continuación:

RUTINA	DESCRIPCIÓN
Chr	Convierte un tipo ordinal en un carácter ANSI.
Ord	Convierte un valor de tipo ordinal al número indicado por su orden.
Round	Convierte un valor de tipo real a un valor de tipo entero, redondeando su valor.
Trunc	Convierte un valor de tipo real a un valor de tipo entero, truncando su valor.
Int	Devuelve la parte entera de su argumento, el cual es de tipo real.
IntToStr	Convierte un número entero en una cadena.
IntToHex	Convierte un número entero a su representación hexadecimal una cadena.
StrToInt	Convierte una cadena en un número entero.
StrToIntDef	Convierte una cadena en un número entero, utilizando un valor por defecto si la cadena no es un número válido.
FloatToStr	Convierte un número real en una cadena.
StrToFloat	Convierte una cadena en un número real.

[6] http://www.programacion.net/

StrToFloatDef	Convierte una cadena en un número real, utilizando un valor por defecto si la cadena no es un número válido.

3.9 Condiciones

Una sentencia condicional se utiliza para ejecutar un grupo de una o más sentencias, dependiendo de una condición de prueba. Existen dos tipos básicos de sentencias condicionales; las sentencias if las sentencias case.

Sentencias if

Existen dos formas de sentencias if:
if...then e if...then...else.

La sintaxis para una sentencia if...then es la siguiente:
if expresión then sentencias

Donde expresión es una condición que devuelve un valor booleano, y sentencias es un grupo de una o más sentencias a ejecutar. Si expresión tiene el valor True, entonces el grupo de sentencias se ejecuta, de otra forma no se ejecuta. Por ejemplo.
if I <> 0 then Result := n/I;

La sintaxis para una sentencia **if...then...else** es:
if expresión then sentencias1 else sentencias2

Donde expresión es una expresión que devuelve un valor booleano, sentencias1 y sentencias2 son grupos de una o más sentencias. Si expresión tiene el valor True, entonces se ejecuta sentencias1, de otra forma se ejecuta sentencias2. Por ejemplo.

```
if num>0 then
   ShowMessage ('El numero es positivo')
else
   ShowMessage ( 'El número es negativo' ) ;
```

O bien:

```
if J<>0  then
  begin
```

```
    Result := I/J;
    Count : = Count + 1;
  end
else
Done := True;
```

Observe que no se debe escribe nunca un punto y coma entre la cláusula then y la palabra else.
En las sentencias if anidadas, algunas sentencias if pueden tener cláusulas else mientras que otras no, pero la sintaxis para los dos tipos de sentencias es la misma. En una serie de condicionales anidadas existen menos cláusulas else que sentencias if, podría ser difícil determinar cuales cláusulas else están unidas a cuales sentencias if. Por ejemplo:

```
if expresión1 then if expresión2 then sentencias1  else sentencias2;
```

Podría parecer que existen dos posibles formas de interpretar la línea anterior:

```
If expresion1 then [ if expresion2 then sentencias1 else sentencias2];
If expresion1 then [ if expresion2 then sentencias1] else sentencias2;
```

El compilador siempre interpreta la línea de la primera forma, es decir, la sentencia:

```
if . . . { expresion1 } then
   if . . . { expresion2 } then
      ... { sentencias1 }
  else
      ... { sentencias2 };
```

es equivalente a:

```
if ... { expresion1 } then
begin
   if . . . { expresion2 } then
      ... { sentencias1 }
   else
      ... { sentencias2 }
end;
```

La regla es que los condicionales anidados se interpretan comenzando con el condicional más anidado, con cada else unido a su if anterior más cercano. Para forzar al compilador a interpretar el código anterior de acuerdo a la segunda opción, se tendría que rescribir de la siguiente manera:

```
if ... { expresion1 } then
begin
    if ... { expresion2 ) then
      ... { sentencias1 }
end
else
       ...{ sentencias2 };
```

3.10 Sentencias Case

La sentencia case proporciona una alternativa más legible a las sentencias if profundamente anidadas, la sintaxis de una sentencia case tiene la siguiente estructura:

```
Case ExpresionOrdinal of
   CasoLista1: sentecias1;
            ...
   casoListaN: senteciasN;
end;
```

Donde ExpresionOrdinal es cualquier expresión de un tipo ordinal y cada casoLista puede ser:

- Un número, una constante declarada, u otra expresión de tipo ordinal compatible con ExpresionOrdinal, que el compilador pueda evaluar sin ejecutar el programa.
- Un subrango de la forma *Primero..Ultimo*, donde *Primero* y *Ultimo* satisfacen el criterio anterior y *Primero* es menor o igual a *Ultimo.*
- Una lista de la forma elemento1, ..., elementoN, donde cada elemento satisface una de las condiciones anteriores.

Cada valor representado por un casoLista debe ser único en una sentencia case, los subrangos y las listas de elementos no se pueden traslapar. Opcionalmente una sentencia case puede tener una cláusula else final. Por Ejemplo:

```
Case ExpresionOrdinal of
   CasoLista1: sentecias1;
            ...
   casoListaN: senteciasN
```

```
  else
     sentencias;
end;
```

Donde sentencias es un grupo de sentencias delimitadas por punto y coma. Cuando se invoca una sentencia case, cuando mucho se ejecutará una sola opción de sentencias1.. .sentenciasN.

El casoLista que posea un valor igual a ExpresionOrdinal determina las sentencias a ejecutar, si ninguno de los casoLista posee el mismo valor que ExpresionOrdinal, entonces se ejecutarán las sentencias de la cláusula else, si existe.

Por ejemplo, la sentecia case:

```
case I of
  1..5: Caption := 'Bajo';
  6..9: Caption := 'Alto';
  0, 10..99: Caption := 'Fuera de Rango';
else
  Caption := '';
end;
```

Es equivalente a la siguiente estructura de if anidados:

```
if I in [1..5] then
  Caption := 'Bajo'
else if I in [6..10] then
    Caption := 'Alto'
    else if (I = 0) or (I in [10..99]) then
      Caption := 'Fuera de Rango'
      else
        Caption := '';
```

3.11 Ciclos

Los ciclos permiten ejecutar un grupo de sentencias repetidamente, utilizando una variable o condición de control para determinar cuando detener el ciclo, Objecí Pascal posee tres tipos de ciclos de control: las sentencias repeat, las sentencias while y las sentencias for.

Sentencias Repeat

La sintaxis de una sentencia repeat es:

repeat sentencia1; ... ; sentenciaN; until expresión;

Donde expresión es una condición que devuelve un valor booleano, y sentencia1;...;sentenciaN es un grupo de sentencias.

Las sentencia repeat ejecuta el grupo de sentencias, comprobando continuamente la expresión después de cada iteración. Cuando expresión devuelve True la sentencia repeat termina. El grupo de sentencias siempre se ejecuta por lo menos una vez, porque expresión no se evalúa sino hasta el final de la iteración.

A continuación se muestra un ejemplo de la sentencia repeat.

```
repeat
  Write (lntroduzca un valor (0..9): ');
  Readln(num);
until (num >= 0) and (num <= 9);
```

Sentencias While

Una sentencia while es similar a una sentencia repeat, excepto que la condición de control se evalúa antes de ejecutar el grupo de sentencias. Por lo tanto, si la condición es falsa, el grupo de sentencias no se ejecuta nunca.

La sintaxis de una sentencia while es la siguiente:

While expresión do sentencias

Donde expresión es una expresión booleana y sentencias es un grupo de una o más sentencias. La sentencia while ejecuta el grupo de sentencias repetidamente, verificando expresión antes de cada iteración. Mientras expresión devuelva True, la ejecución continuará.

A continuación se muestra un ejemplo de sentencia while:

```
while not Eof (Archivo) do
begin
  Readln(Archivo, Línea);
  Process(Línea);
end;
```

Sentencia For

Una sentencia for, a diferencia de una sentencia repeat o una sentencia while, requiere que se especifique el número de iteraciones que se desean ejecutar para un grupo de sentencias.

La sintaxis para una sentencia for es la siguiente:

For contador := ValorInicial to ValorFinal do sentencias

O bien:

For contador := ValorInicial downto ValorFinal do sentencias

Donde contador es una variable local de tipo ordinal, ValorInicial y ValorFinal son expresiones que son compatibles con contador; sentencias es un grupo de una o más sentencias que no modifican el valor de contador. La sentencia for asigna el valor de ValorInicial a contador, entonces ejecuta el grupo de sentencias repetidamente, incrementando o decrementando contador después de cada iteración. La sintaxis for..to incrementa contador, mientras que la sintaxis for..downto lo decrementa.

Cuando contador devuelve el mismo valor que ValorFinal, el grupo de sentencias se ejecuta una vez más y la sentencia for termina. En otras palabras, el grupo de sentencias es ejecutado una vez por cada valor en el rango de ValorInicial hasta ValorFinal. Si ValorInicial es igual a ValorFinal, sentencias se ejecuta exactamente una vez, si ValorInicial es mayor que ValorFinal en una sentencia for..to, o menor que ValorFinal en una sentencia for-.downto, entonces el grupo de sentencias nunca se ejecuta.

A continuación se muestran ejemplos de sentencias for:

```
    for I : = ListBox1.Items.Count - 1 downto 0 do
        ListBox1.Items[I] := UpperCase(ListBoxl.Items[1]);
for I:=1 to 10 do
  for J := 1 to 10 do
  begin
    X :=0;
    For K := 1 to 10 do
      X := X + Mat1[I, K] * Mat2[K, J];
    Mat[I, J]:= X;
  End;
```

3.12 Unidades

"Las unidades son los módulos individuales de código fuente que forman un programa de Object Pascal, toda unidad debe por lo menos constar de las siguientes partes:

- **Una sentencia unit.** Cada unidad debe tener como su primer sentencia una sentencia unit, especificando que el módulo se trata de una unidad e identificándola con un nombre. El nombre de la unidad debe ser idéntico al nombre de archivo que almacena la unidad. Por ejemplo en un archivo llamado Ejemplo.pas la sentencia debe ser: unit Ejemplo;

- **La sección interface**. Después de la sentencia unit, la siguiente línea de código de una unidad debe ser la sentencia interface, toda sentencia posterior, hasta la sentencia implementation, es la información que puede ser compartida con otras unidades del programa. Es en la sección interface de un programa donde se declaran tipos, variables, constantes, procedimientos y funciones que deseamos que sean visibles a otras unidades del programa. En el caso de los procedimientos y funciones solamente se deben incluir declaraciones en la sección interface, nunca cuerpos de procedimientos y funciones.

- **La sección implementation**. Se encuentra después de la sección interface. En esta sección se incluye el cuerpo de los procedimientos y funciones declarados en la sección interface, además en esta sección se declaran tipos, variables y constantes que no se desea que estén disponibles fuera de la unidad donde se declaran.

Opcionalmente, una unidad puede incluir las siguientes secciones:

- **La sección initialization**. Esta sección se encuentra cerca del fin de la unidad y se utiliza para implementar código de initialización, como asignación de recursos que se necesitan antes de que inicie la ejecución del programa, el código dentro de la sección initialization se ejecuta una sola vez.
- **La sección finalization**. Esta sección se encuentra en el fin de la unidad y se utiliza para implementar código de finalización, es decir aquel que tiene como objetivo liberar recursos previamente asignados cuando el programa finaliza."[7]

[7] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

La Cláusula Uses

La cláusula uses es donde se listan las unidades que se desean incluir en un programa o unidad. Por ejemplo, si tenemos un programa llamado MiPrograma que utilizará rutinas y tipos de dos unidades, UnitA y UnitB respectivamente, tendríamos el siguiente código:

Program Miprograma;
Uses UnitA, UnitB;

Las unidades pueden tener dos cláusulas uses, una en la sección interface y otra en la sección implementation. A continuación se muestra el código de una unidad de ejemplo.

Unit Ejemplo;
Interface
Uses Ejemplo2;
Implementation
Uses Ejemplo3;
Initialization
Finalization
End.

Referencias circulares de Unidades

Si se tiene una unidad llamada UnitA que utilice otra unidad llamada UnitB, y a su vez UnitB utilice UnitA, se dice que se tiene una referencia circular de unidades. Lo cual la mayoría de las veces indica un defecto en el diseño de la aplicación, se debe de evitar construir referencias circulares de unidades en una aplicación. La solución a este problema es mover el código que tanto UnitA como UnitB necesitan a una tercera unidad; sin embargo si esto no es posible, se debe mover una de las cláusulas uses a la sección Implementatíon y dejar la otra en la sección interface.

3.13 RTTI (Runtime Type Information)

RTTI es una característica del lenguaje Object Pascal que proporciona a las aplicaciones escritas en Delphi la capacidad de obtener información acerca de los objetos que contiene en tiempo de ejecución. RTTI es la liga entre los componentes de Delphi y su incorporación al IDE.

Todo objeto en Delphi es un descendiente de la clase TObject, por lo tanto, cada objeto contiene un apuntador a su RTTI y dispone de varios métodos que le permiten extraer información útil de la misma. La siguiente tabla muestra algunos de los métodos (rutinas de una clase) que utilizan la RTTI para obtener información de una instancia (objeto) particular de una clase.

MÉTODO	TIPO DE RETORNO	DEVUELVE
ClassName	String	El nombre de la clase del objeto.
ClassType	TCIass	El tipo del objeto.
InheritsFrom	Boolean	Un tipo booleano que indica si la clase Desciende de una clase dada.
ClassParent	TCIass	El tipo del antecesor del objeto.
InstanceSize	word	El tamaño en bytes de una instancia.
Classinfo	Pointer	Un apuntador a la RTTI del objeto en memoria.

Object Pascal proporciona dos operadores, is y as, que permiten realizar comparaciones y forzar tipos de objetos vía RTTI.

El operador as proporciona una manera segura de forzar tipos. Permite forzar el tipo de un objeto a un tipo descendiente y lanza una excepción si la conversión de tipos es inválida.

Por ejemplo: Tenemos un procedimiento al cual podemos pasar como parámetro cualquier tipo de objeto.

procedure Foo(AnObject: TObject);

Si deseamos realizar alguna operación sobre el parámetro AnObject dentro del cuerpo del procedimiento, es probable que tengamos que forzar su tipo a algún tipo descendiente, si suponemos que AnObjet es un descendiente del tipo TEdit, y deseamos modificar el texto que contiene, podemos utilizar el siguiente código.

(AnObject as TEdit).Text := 'Hola Mundo';

Se puede utilizar el operador de comparación booleano is para verificar si dos objetos son de tipos compatibles. Utilice el operador is para comparar un objeto desconocido con un tipo conocido, para determinar que propiedades (estado) , métodos y eventos (comportamiento) podemos asumir que posee el objeto desconocido.

Por ejemplo, podríamos desear verificar si AnObject es compatible con el tipo Tedit antes de intentar forzar su tipo.

If (AnObject is TEdit) then

TEdit(AnObiect).Text : = 'Hola Hundo' ;

Observe que no utilizamos el operador as para realizar el forzado de tipos, debido a que este involucra cierta sobrecarga por causa de su utilización de la RTTI. La primera línea de código ya ha determinado que AnObject es de tipo TEdit, así que se puede optimizar el código realizando un forzado de tipos tradicional en la segunda línea. Un forzado de tipos tradicional no acarrea una sobrecarga en tiempo de ejecución.

3.14 Preguntas y Ejercicio de Repaso

1. Describa las distintas secciones de una unidad de código y el propósito de cada una.

2. ¿Cual de estas sintaxis es correcta en la declaración de variables

a) Var num: Integer; b) num: Double; c)Var num; Integer;

3. Seleccione cuales son los 3 Tipos de Datos predefinidos en Pascal

a)Integer, Float y Double b)Ordinales, Integer y Reales c) Cadenas, Ordinales y Reales

4. ¿Qué características poseen los tipos ordinales?

5. Escribe un pequeño programa donde utilices los dos Tipos de Condiciones

6. Describa la sintaxis de los distintos ciclos en Object Pascal.

7. ¿Cuál es la diferencia entre un procedimiento y una función?

8. Para que sirve la Cláusula Uses

9. ¿Qué es RTTI?

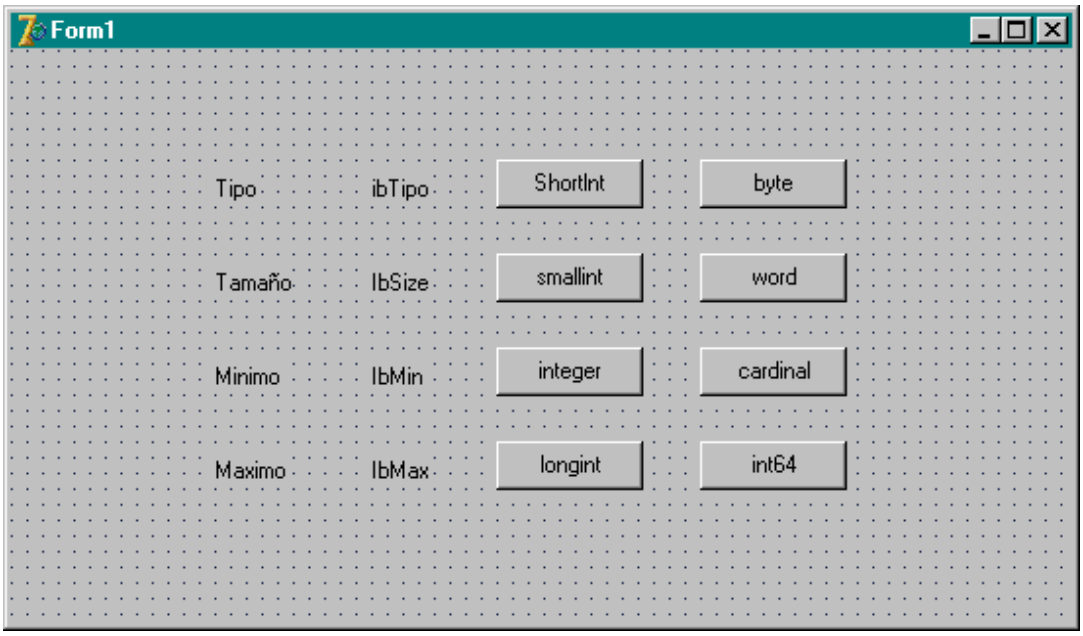
Ejercicio de Repaso

Construiremos una aplicación para mostrar las características de los tipos ordinales de números enteros.

1. Cree en Delphi una nueva forma File │New│Application.
2. Coloque dentro de su forma ocho componentes Tlabel y ocho componentes Tbutton de la pestaña Standard de la paleta de componentes y modifique sus propiedades como se indica a continuación.

COMPONENTE	PROPIEDAD	VALOR
Label	Caption	Tipo:
Label2	Caption	Tamaño:
Label3	Caption	Mínimo:
Label4	Caption	Máximo:
Label5	Name	IbTipo
Label6	Name	IbSize
Label7	Name	IbMin
LabeI8	Name	IbMax
Button1	Name Caption Tag	BtnShortInt Shortint 0
Button2	Name Caption Tag	BtnSmallInt Smallint 1
Button3	Name Caption Tag	Btninteger Integer 2
Button4	Name Caption Tag	BtnLongInt Longint 3
Button5	Name Caption Tag	BtnByte Byte 4
Button6	Name Caption Tag	BtnWord Word 5
Button7	Name Caption Tag	BtnCardinal Cardinal 6
Button8	Name Caption Tag	Btnlnf64 lnt64 7

Su pantalla debe tener el siguiente aspecto:

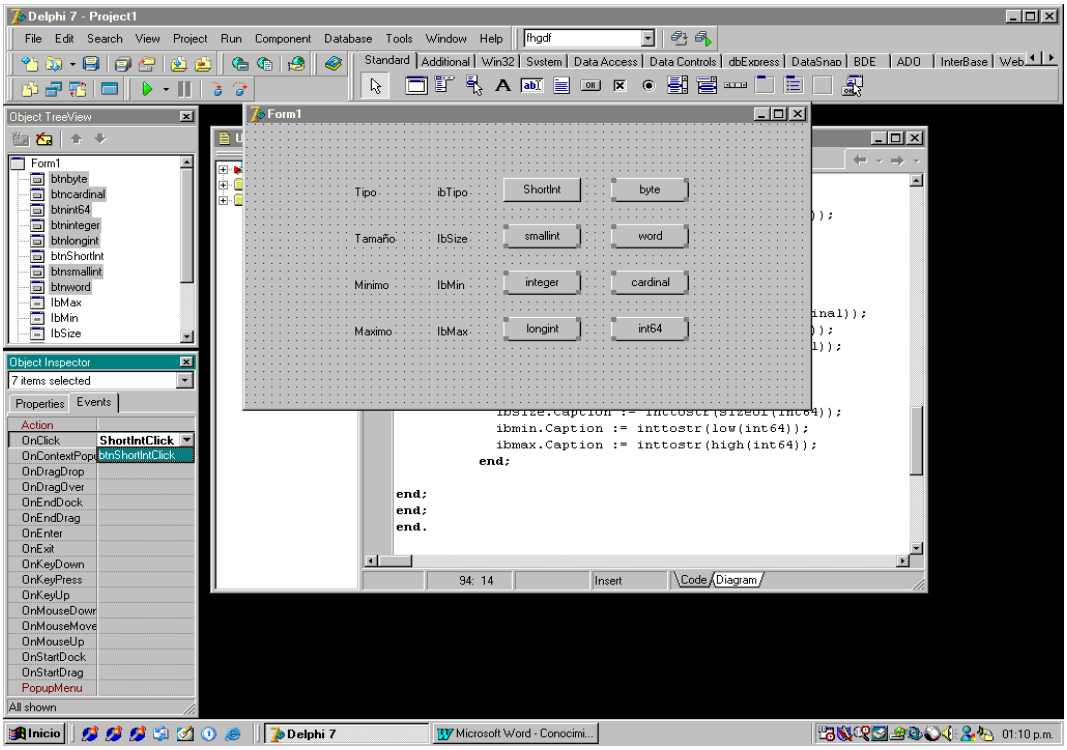


3. Implemente el siguiente código en el evento Onclick del primer botón btnShortlnt.

```
procedure TForm1.btnShortlntClick(Sender: TObject);
begin
  case (sender as Tbutton).Tag of
     0: begin
        ibtipo.Caption := 'Shortint';
        ibsize.Caption := inttostr(sizeof(shortint));
        ibmin.Caption := inttostr(low(shortint));
        ibmax.Caption := inttostr(high(shortint));
      end;
    1: begin
        ibtipo.Caption := 'Smallint';
        ibsize.Caption := inttostr(sizeof(smallint));
        ibmin.Caption := inttostr(low(smallint));
        ibmax.Caption := inttostr(high(smallint));
      end;
     2: begin
        ibtipo.Caption := 'Integer';
        ibsize.Caption := inttostr(sizeof(integer));
```

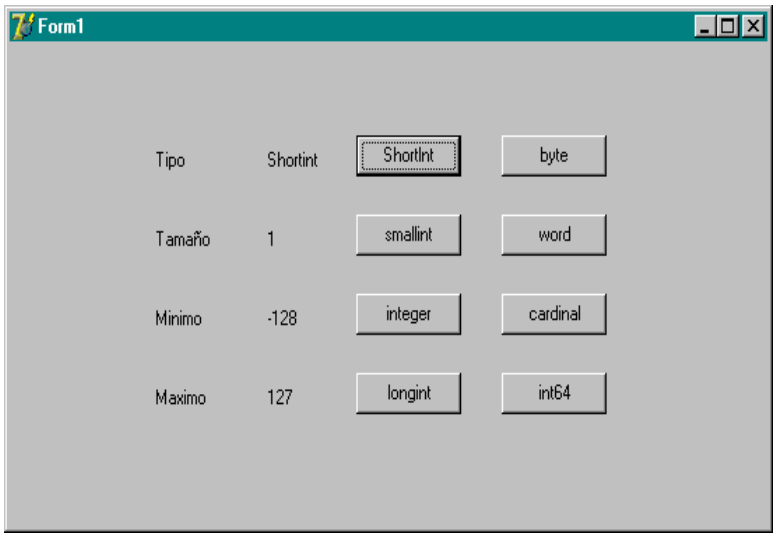
```
        ibmin.Caption := inttostr(low(integer));
        ibmax.Caption := inttostr(high(integer));
      end;
    3: begin
        ibtipo.Caption := 'LongInt';
        ibsize.Caption := inttostr(sizeof(longint));
        ibmin.Caption := inttostr(low(longint));
        ibmax.Caption := inttostr(high(longint));
      end;
    4: begin
        ibtipo.Caption := 'Byte';
        ibsize.Caption := inttostr(sizeof(byte));
        ibmin.Caption := inttostr(low(byte));
        ibmax.Caption := inttostr(high(byte));
      end;
    5: begin
        ibtipo.Caption := 'Word';
        ibsize.Caption := inttostr(sizeof(word));
        ibmin.Caption := inttostr(low(word));
        ibmax.Caption := inttostr(high(word));
      end;
    6: begin
        ibtipo.Caption := 'Cardinal';
        ibsize.Caption := inttostr(sizeof(cardinal));
        ibmin.Caption := inttostr(low(cardinal));
        ibmax.Caption := inttostr(high(cardinal));
      end;
   7: begin
        ibtipo.Caption := 'Int64';
        ibsize.Caption := inttostr(sizeof(int64));
        ibmin.Caption := inttostr(low(int64));
        ibmax.Caption := inttostr(high(int64));
      end;
end;
end;
end.
```

4. Delphi nos permite ligar un manejador de eventos a más de un componente, seleccione los botones restantes pulsando click sobre ellos mientras mantiene presionada la tecla Shift y en el inspector de objetos seleccione para el evento Onclick el manejador del primer botón, como se muestra en la siguiente figura.



El manejador de eventos primero determina el valor de la propiedad Tag del botón que lo ejecuta, y dependiendo de este valor ejecuta alguno de los casos de la sentencia case, mostrando para el tipo ordinal seleccionado su nombre, tamaño y valores mínimo y máximo del rango del tipo.

5. Ejecute la aplicación (F9).



IV. Programación Orientada a Objetos

4.1 Introducción

"Todos los lenguajes de programación proveen un cierto nivel de abstracción, por ejemplo el lenguaje ensamblador representa una abstracción de la máquina donde se ejecuta, muchos lenguajes imperativos que le siguieron (FORTRAN, BASIC, C) fueron abstracciones del lenguaje ensamblador, pero aún requerían que los programadores pensaran en términos de la estructura de la computadora en lugar de pensar en términos del problema que se deseaba resolver.

La razón porque los programas que emplean un lenguaje imperativo sean costosos, difíciles de escribir y mantener es debido a que el programador debe establecer una relación mental entre el modelo de la máquina (en el "espacio de la solución") y el modelo del problema (en el "espacio del problema").

La Programación Orientada a Objetos va un paso más allá, modelando el problema que se trata de resolver, proporcionando al programador herramientas que representan elementos (objetos) en el espacio del problema. De esta forma proporciona una abstracción más flexible y poderosa de la que se tenía antes.

Por lo tanto la POO permite describir el problema en términos del problema, en lugar de en términos de la solución, aunque todavía existe una conexión con la computadora, cada objeto es como una pequeña computadora, en el sentido de que posee un estado y operaciones que puede ejecutar, sí pensamos en los objetos del mundo real, veremos que todos tienen características y un comportamiento.

Delphi es un lenguaje orientado a objetos, lo cual significa que soporta las cuatro características básicas de todo lenguaje orientado a objetos: abstracción, encapsulamiento, herencia y polimorfismo."[1]

4.2 El modelo de Objetos de Delphi

Clases

Una declaración de clase es una declaración de tipo, la cual describe los campos (variables), métodos (procedimientos y funciones) y propiedades de la clase, además posee una o más secciones para los diferentes niveles de acceso. Dentro de cada sección, se pueden declarar cualquier número de campos, seguidos por métodos y declaraciones de propiedades. Los métodos y propiedades pueden intercalarse, pero todos los campos deben preceder a los métodos y propiedades en cada sección. Además mediante el mecanismo de la herencia una clase puede heredar de otra sus campos, métodos y propiedades.

[1] Marco Cantú, "MASTERING DELPHI 7" Sybex, USA, 2003 pag.43-55

En Delphi, una clase tiene una sola clase base, de la cual hereda todos sus campos, propiedades y métodos. Si no se especifica explícitamente una clase base, Delphi utiliza TObject.

Una declaración de clase es una declaración de tipo que comienza con la palabra reservada class. Esta declaración contiene los campos métodos y propiedades de la clase y termina con la palabra reservada end. Cada declaración de método es un encabezado, la implementación del mismo se debe de incluir en la misma unidad (con la posible excepción de métodos abstractos). En Delphi los nombres de las clases comienzan con la letra "T", como en TObject.

El Modelo de Objetos de Delphi es similar al de otros lenguajes orientados a objetos como Java y C++. La siguiente tabla presenta una comparación entre Delphi y los citados lenguajes.

CARACTERISTICAS DEL LENGUAJE	DELPHI	JAVA	C++	V. BASIC
Herencia	✓	✓	✓	
Herencia Múltiple			✓	
Interfaces	✓	✓		✓
Una sola clase base	✓	✓		✓
Metaclases	✓	✓		
Métodos Virtuales	✓	✓	✓	
Métodos de Clase (estáticos)	✓	✓	✓	
Recolección de basura		✓		
Tipos Variant	✓			✓
Verificación de tipos estática	✓	✓	✓	
Manejo de Excepciones	✓	✓	✓	✓
Sobrecarga de Funciones	✓	✓	✓	
Sobrecarga de operadores			✓	
Funciones estándar (no de clase)	✓		✓	✓
Variables estándar (no de objeto)	✓		✓	✓
Propiedades	✓			✓
RTTI (Información en Tiempo de Ejecución)	✓	✓		
Tipos Genéricos (templates)			✓	
Paso de Mensajes	✓			
Ensamblador Integrado	✓			
Funciones en línea			✓	

Una clase puede también implementar cualquier número de interfaces. Por lo tanto el modelo de objetos de Delphi se asemeja al de Java, donde una clase puede extender una sola clase e implementar muchas interfaces.

4.2.1Visibilidad de los Miembros de una Clase.

Delphi ofrece control sobre el comportamiento de un objeto al permitir declarar los campos y métodos de una clase con directivas como private, protected, public, published y automated. La sintaxis para utilizar estas palabras reservadas es la siguiente:

```
TEjemplo = class
  private
UnCampoPrivado: Integer;
    OtroCampoPrivado: Boolean;
 protected
     procedure UnProcedimientoProtegido;
fuction UnaFuncionProtegida: Byte;
public
    constructor Create;
    destructor Destroy;
published
property UnaPropiedad read UnCampoPrivado write UnCampoPrivado;
end;
```

Se pueden declarar cualquier número de campos y métodos dentro de cada directiva.

“private

Estas partes del objeto son solo accesibles al código en la misma unidad donde se implementa el objeto. Se utiliza para ocultar los detalles de implementación de los usuarios de la clase.

protected

Los miembros protegidos de un objeto pueden ser accedidos por descendientes del objeto. Esta capacidad permite ocultar los detalles de la implementación de los usuarios mientras se provee de máxima flexibilidad a los descendientes del objeto.

public

Estos campos y métodos son accesibles en cualquier parte del programa. Los constructores y destructores deben ser públicos.

published

La sección publicada genera la Información en Tiempo de Ejecución (RTTI) del objeto, la cual permite a otras partes de una aplicación obtener información del mismo. El Object Inspector utiliza la RTTI para construir su lista de propiedades.

Automated

Los miembros de una clase en esta sección tienen la misma visibilidad que los miembros públicos. La diferencia es que información de tipo Automation es generada para los miembros en esta sección, los cuales aparecen generalmente solo en clases de Windows y no se recomiendan para programación en Linux. La palabra reservada automated es mantenida por razones de compatibilidad con Delphi 2."[2]

4.3 Objetos

Un objeto es una instancia dinámica de una clase, la cual contiene valores para todos los campos declarados en la clase y sus clases ancestro. Un objeto también contiene un campo oculto que almacene una referencia a la clase del objeto. En otras palabras un objeto es el espacio de memoria donde Delphi almacena los valores de todos los campos del objeto y una referencia de objeto es un apuntador a dicho objeto, la única manera de utilizar un objeto en Delphi es a través de una referencia de objeto la cual toma la forma de una variable, una función o una propiedad.

La forma más común de utilizar una referencia de clase es en la creación de instancias de una clase a través de un constructor de clase, pero también se puede utilizar una referencia de clase para probar el tipo de un objeto (con el operador is) o para forzar el tipo de un objeto a un tipo particular (con el operador as).

Para verificar el tipo de una clase se utiliza el operador is, el cual devuelve True si la referencia de clase es la clase del objeto o cualquiera de sus clases ancestro. De otro modo devuelve False si la referencia de objeto es nil o los tipos no concuerdan, por ejemplo:
if TEmpleado is TPersona then . . .
if TEmpleado is TObject then...

Se puede realizar un cast para obtener una referencia de objeto con un tipo diferente, un cast no modifica un objeto, solamente proporciona una nueva referencia de objeto. El operador as es precisamente el utilizado para realizar un casi sobre un objeto, este operador verifica el tipo del objeto y levanta una excepción (de tipo EInvalidCast) si la clase del objeto no es un descendiente de la clase destino.

[2] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

El siguiente código ejemplifica el uso del operador as:
// Si Objeto no es descendiente de TPersona se levantará una excepción with Objeto as Tpersona do…

4.3.1 Constructores.

Un constructor es un método especial que crea e inicializa una instancia de una clase. La declaración de un constructor debe comenzar con la palabra reservada constructor.
Por ejemplo:
constructor Create;
constructor Create(AOwner: TComponent);

La mayoría de los constructores se llaman Create, aunque es válido que una clase tenga constructores con nombres diferentes esto no se recomienda.

Para crear un objeto, hay que llamar al método constructor del tipo de clase. El desarrollador es responsable de crear al objeto mediante un constructor del tipo de clase. Por ejemplo;
MiObjeto := TMiClase.Create;

Esto asigna espacio para el nuevo objeto en el heap, coloca los valores para todos los campos ordinales en cero, asigna nil a todos los apuntadores y campos de tipo de clase, y hace que todas las cadenas sean vacias. Otras acciones especificadas en la implementación del constructor se ejecutan a continuación, típicamente, se inicializan objetos con los valores pasados en los parámetros del constructor. Finalmente, el constructor devuelve una referencia al objeto asignado e inicializado. El tipo del valor devuelto es el mismo que el especificado en la llamada al constructor.

La primera acción de un constructor es usualmente llamar un constructor heredado de la clase base para inicializar los campos heredados del objeto. El constructor entonces inicializa los campos introducidos en la clase descendiente.

Para liberar un objeto una vez que no se necesita, se debe llamar a! método Free, una práctica común consiste en colocar el método Free en un bloque try-finally para asegurarse de que el objeto se libere aunque ocurra una excepción. Por ejemplo:

```
//Invocación del constructor de clase
Objeto := Tejemplo.Create;
Try
   ...
   //Código que utiliza el objeto
```

```
finally
   //Liberación del objeto
   Objeto.Free;
end;
```

Cada objeto posee una copia de todos sus campos. Un campo no puede ser compartido a través de instancias de la misma clase.

Un objeto siempre se instancia dinámicamente desde el heap, por lo tanto una referencia a un objeto es como un apuntador (pero sin la sintaxis usual de los apuntadores de Pascal). Cuando se asigna una referencia de objeto a una variable, Delphi tan solo copia el apuntador, no el objeto entero. Cuando un objeto ya no se necesita es responsabilidad del desarrollador liberar explícitamente dicho objeto pues Delphi carece de un mecanismo de recolección de basura.

4.3.2 Destructores.

Un destructor es un método especial que destruye el objeto que lo invocó y libera su memoria, un destructor tiene un parámetro oculto. La primera llamada al destructor pasa el parámetro con un valor de True, esto le dice a Delphi que llame FreeInstance para liberar el objeto. Si el destructor llama un destructor heredado. Delphi pasa el parámetro con un valor de False para prevenir que el destructor heredado trate de liberar el mismo objeto. Por ejemplo:

```
Destructor Destroy;
Destructor Destroy; override;
```

Aunque una clase puede tener más de un destructor, se recomienda que cada clase redeclare el método heredado Destroy y que no declare otros destructores.

Un destructor se debe invocar desde una referencia de objeto. Por ejemplo:

```
MiObjeto.Destroy;
```

Antes de que Delphi ejecute el destructor, llama el método virtual BeforeDestruction. Se puede redeclarar BeforeDestruction para colocar código que deba de ejecutarse antes que el destructor.

Al declarar una clase, se puede redeclarar el método Destroy, pero no se debe redeclarar el método Free. Cuando se libera un objeto, se debe de llamar el método Free y no al destructor, porque Free primero verifica si la referencia de objeto tiene un valor nil (nulo) y llama Destroy solo para referencias de objeto no nulas.

Cuando se llama al destructor, primero se ejecutan las acciones en la implementación del destructor, después el almacenamiento que utilizaba el objeto es liberado.

El siguiente es un ejemplo de la implementación del destructor para la clase TImage:

```
destructor TImage.Destroy;
begin
   FPicture. Free;
   Inherited Destroy;
End;
```

La última acción en al implementación de un destructor es por lo general una llamada al destructor heredado para destruir los campos heredados del objeto.

Cuando ocurre una excepción durante la creación de un objeto, se llama automáticamente a Destroy para liberar al objeto inacabado. Por lo tanto, el Destructor debe estar preparado para liberar objetos parcialmente construidos.

4.4 Abstracción

La abstracción es un mecanismo poderoso que permite aislar las características esenciales de un objeto ignorando aquellos aspectos que no intervienen en el problema que se desea resolver.

Todo conocimiento se halla necesariamente unido a procesos de abstracción; en la Programación Orientada a Objetos este proceso permite ignorar los detalles de implementación de los objetos, centrándose en las características (propiedades) y comportamiento (métodos) de los mismos, además de cómo se relacionan entre ellos.

4.5 Encapsulamiento

La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se centra en el comportamiento visible de un objeto, mientras el encapsulamiento se centra en la implementación que da lugar a este comportamiento. El encapsulamiento se consigue a menudo mediante la ocultación de información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; típicamente, la estructura de un objeto esta oculta, así como la implantación de sus métodos.

El encapsulamiento proporciona barreras explícitas entre abstracciones diferentes y por tanto conduce a una clara separación de intereses. Por ejemplo al diseñar una aplicación de bases de datos, es práctica común el escribir programas de forma que no se preocupen de la representación física de los datos, sino que dependan sólo de un

esquema que denota la vista lógica de los mismos. En ambos casos, los objetos a un nivel de abstracción están protegidos de los detalles de implementación a niveles más bajos de abstracción.

Cada clase debe tener dos partes: una interfaz y una implementación. La interfaz de una clase captura solo su vista externa, abarcando la abstracción que se ha hecho del comportamiento común de todas las instancias de la clase. La implementación de una clase comprende la representación de la abstracción así como los mecanismos que consiguen el comportamiento deseado. La interfaz de una clase es el único lugar en el que se declaran todas las suposiciones que un cliente puede hacer acerca de todas las instancias de la clase; la implementación encapsula detalles acerca de los cuales ningún cliente puede realizar suposiciones.

En la Programación Orientada a Objetos, los objetos interactúan unos con otros por medio de mensajes. La única información que un objeto posee acerca de otro objeto es la interfaz del objeto. De esta forma los datos y la lógica del objeto se ocultan de otros objetos.

Esto permite a un desarrollador separar la implementación de un objeto de su comportamiento. Esta separación crea un efecto de "caja negra" donde el usuario está aislado de cambios en la implementación. Mientras que la interfaz permanezca inmutable, cualquier cambio a la implementación interna es transparente para el usuario.

En resumen:
"El encapsulamiento es el proceso de almacenar en un mismo compartimiento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar la interfaz contractual de una abstracción y su implementación".

4.6 Herencia

La herencia es la capacidad de crear objetos nuevos a partir de otros objetos. En otras palabras la herencia es la capacidad de un objeto para utilizar la funcionalidad prevista en clases ascendientes. El objetivo final de este mecanismo es la reutilización de código anteriormente desarrollado.

La herencia supone una dase base y una jerarquía de clases que contiene las clases derivadas de la clase base. Las clases derivadas pueden heredar el código y los datos de su clase base y añadir su propio código para proveer más funcionalidad.

Definimos una clase de objetos como un conjunto de objetos que comparten características y un comportamiento común, los cuales se definen en una clase base. Las clases derivadas se crean en un proceso de definición de nuevos tipos reutilizando el código anteriormente desarrollado en la definición de la clase base. Este proceso se

denomina programación por herencia. Las clases que heredan propiedades de una clase base pueden a su vez servir como definiciones base de otras clases, de esta manera se consigue extender la jerarquía de clases.

Semánticamente, la herencia denota una relación "es un". Por ejemplo un gato "es un" tipo de mamífero, el algoritmo de la burbuja "es un" tipo de algoritmo de ordenación, etc. Así la herencia implica una jerarquía de generalización/especialización, en la que una subclase especializa el comportamiento más general de sus clases ascendientes. Realmente, este es el concepto fundamental de la herencia: Si el objeto x "no es" un tipo de y, entonces x no debería heredar de y.

A medida que se desarrolla la jerarquía de clases, la estructura y comportamiento comunes a diferentes clases tenderá a migrar hacia clases ascendientes comunes. Las clases ascendientes representan abstracciones generalizadas y las clases derivadas representan especializaciones en las que los datos y los métodos de las clases ascendientes, sufren añadidos, modificaciones e incluso ocultaciones.

La VCL de Delphi es un ejemplo de una jerarquía de clases donde la clase base es la clase TObject.

A diferencia de otros lenguajes de programación como C++, Object Pascal no soporta herencia múltiple. La herencia múltiple representa el concepto de un objeto derivado de dos clases diferentes, creando un objeto que contiene el código y los datos de dos objetos ascendentes. Aunque esta funcionalidad parezca útil, a menudo introduce más problemas e ineficiencia de los que resuelve.

En Delphi, cada clase hereda de una sola clase raíz, TObject, si no se especifica explícitamente una clase base, Delphi implícitamente utiliza TObject como la clase base.

4.7 Polimorfismo

"El polimorfismo es la propiedad que indica, literalmente, la posibilidad de que una entidad tome muchas formas. Además el polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, dependiendo del tipo del objeto al cual se hace referencia en un momento dado"[3].

Por ejemplo, si describimos una clase *mamíferos* podemos observar que la operación comer es realizada por todas las instancias de la clase ya que todo mamífero debe ser capaz de realizar dicha operación; sin embargo la manera de llevar a cabo dicha operación variará por ejemplo entre un herbívoro y un carnívoro.

[3] http://www.delphimania.freeservers.com/DimeQueEs/N-Z.html

El polimorfismo indica la posibilidad de tomar un objeto el cual sea una instancia del tipo mamífero e indicarle que ejecute la operación comer, esta acción se llevará a cabo de diferente forma, según sea el tipo específico de mamífero sobre el que se aplica.

Las clases, la herencia y el polimorfismo son aspectos claves en la Programación Orientada a Objetos. El polimorfismo desarrolla su máximo potencial en la derivación de clases, mediante el mecanismo de la herencia. Un ejemplo clásico consiste en definir en una clase genérica *figura* que acepte mensajes como *dibujar, borrar y mover*. Cualquier clase derivada de figura es un tipo de *figura* que puede recibir los mismos
mensajes. Al enviar un mensaje dibujar, este se realizaría de manera distinta según la clase derivada específica (círculo, triángulo, etc.).

4.8 Preguntas de Repaso

1. Describa cuales son las características básicas de todo lenguaje Orientado a Objetos.

2. Subraya todas las caracteristicas que correspondan al Delphi.

Herencia Multiple, Interfaces, Métodos Virtuales, Recolección de basura, Ensamblador Integrado

3. Describa el mecanismo de la herencia.

4. Describa la finalidad de un constructor y escribe su sintaxis.

5. Mencione la finalidad de un destructor y escribe su sintaxis.

6. ¿Cómo se relacionan los Objetos y las Clases?

7. ¿Por qué se dice que Abstracción y Encapsulamiento son complementarios?

8. Describa las directivas de Accesibilidad de los miembros de una clase en Delphi.

9. ¿Cuál es la función del polimofirmo?

V. La Biblioteca de Componentes Visuales (VCL/CLX)

5.1 Jerarquías de Clases

"El diseño de sistemas de software es una actividad compleja, debido entre otras causas a que cada vez el hardware es más poderoso y los usuarios esperan que el software aproveche las capacidades del hardware y se libere cada vez más rápido. Uno de los principales problemas a los que se deben de enfrentar los desarrolladores de aplicaciones es el manejo de la complejidad, para lo cual se han diseñado diversas metodologías de desarrollo, siendo las metodologías orientadas a objetos poderosas herramientas para lograr este fin.

Al analizar un sistema complejo podemos dividirlo en ciertos objetos interrelacionados, cada uno cumpliendo un rol en el sistema, de forma que forman jerarquías de diferentes tipos de objetos llamadas clases, las cuales son estructuras de objetos que colaboran entre sí a través de mecanismos bien definidos.

De esta forma, aunque se puede tratar como distinta cada Instancia de un determinado tipo de objeto, puede asumirse que comparte las mismas propiedades y comportamiento que todas las demás instancias de su tipo de objeto.

El reconocimiento de las jerarquías en un sistema de software complejo no es usualmente una tarea trivial, debido a que se requiere descubrir patrones entre muchos objetos, cada uno de los cuales puede presentar un comportamiento complejo. Sin embargo, una vez que se han descubierto estas jerarquías, se simplifica la comprensión del sistema complejo.

Delphi proporciona una jerarquía de clases de objetos llamada la VCL (Visual Component Library), Kylix (la versión de Delphi para Linux) introdujo una nueva jerarquía de clases llamada CLX (Component Library for Crosss Platform). Delphi 7 incluye ambas jerarquías de clases. Para el desarrollo de aplicaciones visuales las dos jerarquías son una alternativa una de la otra, pero ambas comparten las clases básicas."[1]

5.2 La VCL

Aunque a la VCL se le llama la Biblioteca de Componentes Visuales, no todas sus clases son visuales en tiempo de ejecución, ni todas sus clases son componentes, un componente es un tipo especial de objeto, el cual es una instancia de una clase que debe descender de la clase TComponent, la cual a su vez desciende de la clase TPersistent.

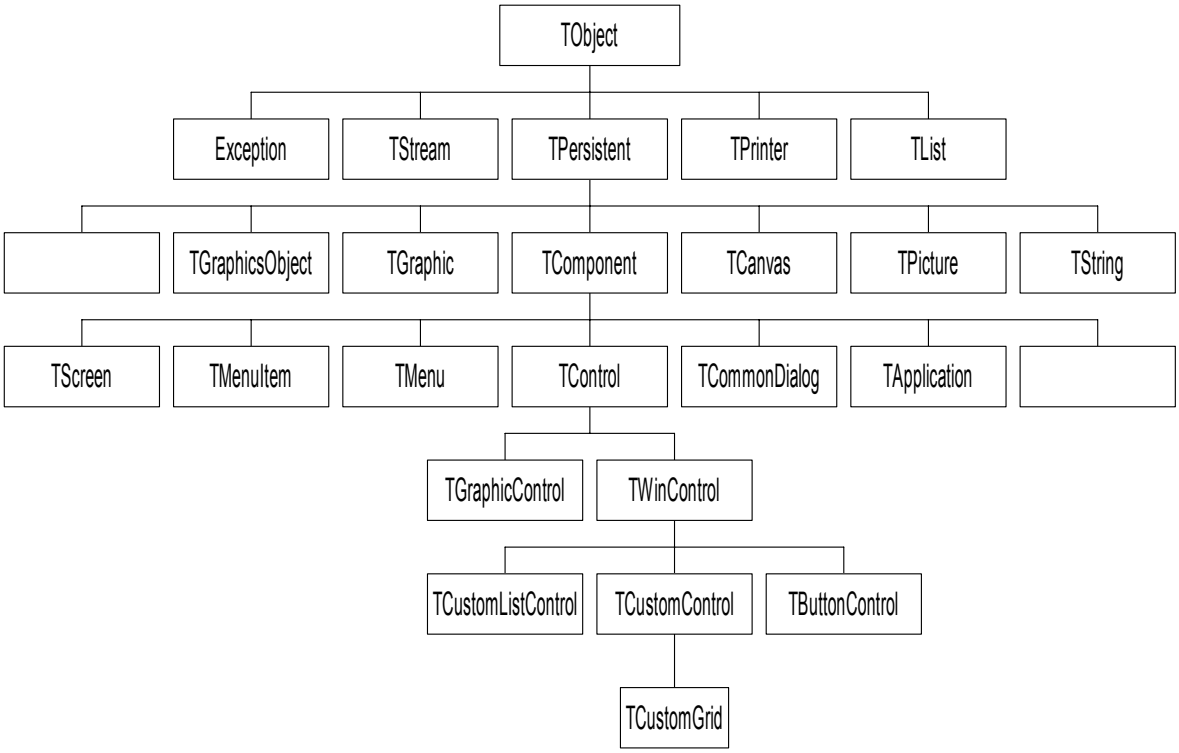
[1] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

Las clases que no descienden de TComponent no están disponibles a través de la Paleta de Componentes, y no pueden colocarse sobre una forma, pero pueden ser manipuladas visualmente por medio del Inspector de Objetos, como subpropiedades de otras propiedades, o como elementos en conjuntos de varios tipos. Así que aún las clases que no representan componentes se utilizan frecuentemente cuando se desarrollan aplicaciones en el IDE.
Las clases de componentes se pueden dividir en dos grupos: los controles y los componentes no visuales.

Controles. Las clases de controles descienden de la clase TControl, y poseen diversas propiedades y métodos que les permiten tener una posición y tamaño en pantalla.

Componentes no visuales. Son las clases de componentes que no son controles. En tiempo de diseño un componente no visual aparece con la forma de un icono. En tiempo de ejecución estos componentes son no visibles, como el componente TSQLConnection, el cual representa una conexión a una base de datos.

La siguiente figura muestra algunas de las principales clases de la VCL.



A continuación describiremos algunas de las clases más importantes de la VCL.

5.3 TObject

TObject es la clase raíz de la Jerarquía de clases de la VCL, esto significa que toda clase tiene como ancestro común a TObject.

Esta clase encapsula el comportamiento fundamental de todas las clases de la VCL, incluyendo los métodos de la siguiente tabla, muchos de los cuales son utilizados internamente por el IDE y no deberían invocarse directamente, salvo en circunstancias especiales. Otros métodos son reintroducidos en clases descendientes que poseen un comportamiento más especializado.

METODO	DESCRIPCION
AfterConstruction	Este método responde después de que se ha ejecutado el constructor.
BeforeDestruction	Este método responde justo antes de que se ejecute el primer destructor.
Classinfo	Devuelve un apuntador a la tabla de RTTI del tipo del objeto.
ClassName	Devuelve una cadena indicando el tipo de instancia del objeto.
ClassNameIs	Determina si un objeto es de un tipo específico.
ClassParent	Devuelve el tipo del ancestro inmediato de la clase.
ClassType	Devuelve la referencia de clase de la clase del objeto.
Cleanupinstance	Realiza la finalización de cadenas largas, variants y variables de interfaz de una clase.
Create	Construye un objeto e inicializa su información.
DefaultHandler	Proporciona la interfaz a un método que procesa registros de mensajes.
Destroy	Dispone de una instancia de objeto.
Dispatch	Llama a métodos manejadores de objetos del objeto, basado en el contenido de su mensaje parámetro.
FieldAddress	Devuelve la dirección de un campo publicado de! Objeto.
Free	Destruye un objeto y libera su memoria asociada si es necesario.
Freeinstance	Libera la memoria asignada por una llamada previa al método Newinstance.
Getinterface	Devuelve una interfaz especificada.
GetInterfaceEntry	Devuelve la entrada a una interfaz especificada implementada en una clase.
GetInterfaceTable	Devuelve un apuntador a la estructura que contiene todas las interfaces implementadas por una clase dada.
InheritsFrom	Determina si una clase dada desciende de una segunda clase.
Initinstance	Inicializa una nueva instancia de objeto a ceros e inicializa la tabla de apuntadores a métodos virtuales.

InstanceSize	Devuelve el tamaño en bytes de cada instancia del tipo del objeto.
MethodAddress	Devuelve la dirección de un método publicado.
MethodName	Devuelve una cadena que contiene el nombre del método localizado en una dirección especificada.
NewInstance	Asigna memoria para una instancia del tipo del objeto y devuelve un apuntador a dicha instancia.
SafeCallException	Maneja las excepciones en los métodos declarados utilizando la convención de llamadas seguras.

5.4 TPersistent

"TPersistent desciende de TObject, y es la clase base que encapsula el comportamiento de todos los objetos que pueden ser asignados a otros objetos y que pueden leer y escribir sus propiedades hacia y desde un stream, un stream es una clase que permite leer y escribir información desde diferentes medios como memoria, cadenas, sockets y campos BLOB de bases de datos."[2]

La clase TPersistent define los métodos descritos en la siguiente tabla:

METODO	DESCRIPCION
Assign	Copia el contenido de otro objeto similar.
AssignTo	Copia las propiedades de un objeto a un objeto de destino.
DefineProperties	Proporciona una interfaz para un método para leer y escribir información no publicada.
Destroy	Destruye la instancia de TPersistent y libera su Memoria.
GetNamePath	Devuelve el nombre del objeto así como aparece en el Inspector de Objetos.
GetOwner	Devuelve el Owner de un objeto.

Como su nombre implica, esta clase maneja persistencia, la cual es un elemento clave en el desarrollo visual. En tiempo de diseño en Delphi se manipulan componentes, los cuales se salvan en archivos .DFM, y estos se recrean en tiempo de ejecución cuando la forma se crea.

De hecho el mecanismo de manejo de streams no lo implementa la clase TPersistent, sino sus clases descendientes. En otras palabras, solamente es posible utilizar la persistencia con clases que descienden de TPersistent.

[2] Marco Cantú, "MASTERING DELPHI 7" Sybex, USA, 2003 pag.112

De los métodos de la clase TPersistent, uno de los más útiles es el método Assign, el cual se puede utilizar para copiar el valor actual de un objeto. En la VCL, este método es implementado por muchas clases que no son componentes, pero por muy pocas clases de componentes. Muchas clases reimplementan el método Assign.

5.5 TComponent

Cuando se coloca un componente en una forma, el Diseñador de Formas de Delphi crea una instancia del componente, creando un nuevo objeto de la clase correspondiente, sin embargo un componente no está definido como cualquier declaración de clase, todos los componentes están definidos por una clase descendiente de TComponent.

La clase TComponent proporciona la funcionalidad básica que requiere un componente, por ejemplo TComponent implementa las propiedades Name y Tag que heredan todos los componentes, además proporciona los métodos y propiedades que permiten a un componente ser manipulado en Tiempo de Diseño en el Diseñador de Formas.

A continuación se en listan las propiedades de la clase TComponent.

PROPIEDAD	DESCRIPCION
ComObject	Especifica la referencia a una interfaz implementada por el componente.
ComponentCount	Indica el número de componentes de los que es dueño el componente.
ComponentIndex	Indica la posición del componente en el arreglo de componentes de su Owner.
Components	Lista todos los componentes de los que el componente es dueño.
ComponentState	Describe el estado actual del componente, indicando cuando un componente necesita evitar ciertas acciones.
ComponentStyIe	Determina el comportamiento del componente.
Designinfo	Contiene información utilizada por el diseñador de formas.
Name	Indica el nombre del componente.
Owner	Indica al componente responsable de manejar en un stream y liberar el componente.
Tag	Almacena un valor entero dentro del componente.
VCLComObject	Representa información utilizada internamente por componentes que soportan COM.

5.6 Preguntas de Repaso

1. Describa qué es la VCL y la CLX

2. Subraya el nombre de Delphi para Linux

a) Onwer b) Kylix c)TObject

3. ¿Cuál es la clase base de la VCL?

4. Subraya los dos grupos en que se divide la clase de componentes

a) Kylix y Delphi b)VCL y CLX c)Controles y componetes visuales

5. Subraya cuales de estos metodos corresponden a Tobject

a) Updated y Create b) Freeinstance y Cleanupinstance c) ValidateRename y Notification

6. Menciona 5 clases que descienden de TObject

7. La clase TPersistent cual es el metodo mas usado

8. ¿De qué clase descienden todos los componentes de la VCL?

VI. Administración de Proyectos

6.1 La Estructura de un Proyecto de Delphi

Para poder construir y administrar proyectos en Delphi7 es necesario conocer la arquitectura de los elementos que forman un proyecto.

Un proyecto es un conjunto de archivos que forman una aplicación o una librería de ligado dinámico (DLL). Los proyectos se pueden agruparen un grupo de proyectos, el cual permite organizar y trabajar con proyectos relacionados, como aplicaciones y DLL's que trabajan juntas o partes de una aplicación multi-capas.

Un proyecto de Delphi7 está formado por varios archivos relacionados, algunos de ellos son creados en tiempo de diseño conforme se definen formas. Otros no son creados sino hasta que se compila el proyecto. A continuación describiremos los distintos tipos de archivos que forman un proyecto en Delphi7.

6.1.1 El Archivo del Proyecto

El archivo principal del proyecto se genera en tiempo de diseño y tiene la extensión .dpr. Este es el archivo en donde la forma principal y todas las formas que se crean automáticamente son instanciadas. Normalmente no es necesario editar manualmente este archivo, excepto para rutinas especiales de inicialización.

Para ver el código fuente de! archivo de un proyecto en Delphi seleccione el menú Project | View Source. El contenido mínimo de éste archivo se muestra en el siguiente listado:

```
Program Project1;
uses
   Forms,
   Unitl in 'Unitl.pas' {Form1};
{$R *.res}
begin
application.Initialize;
Application.CreateForm(Tform1, Form1) ;
Application.Run;
end.
```

Note que el archivo lista la unidad Unitl en la cláusula uses. Los archivos de proyecto listan todas las unidades de formas que pertenecen al proyecto.

La línea {$R *.res} se refiere al archivo de recursos del proyecto, indicando al compilador que ligue este archivo de recursos, el cual debe de tener el mismo nombre que el archivo del proyecto con una extensión .res. El archivo de recursos del proyecto contiene el icono del proyecto e información de la versión del mismo.

El bloque begin..end es donde se encuentra el código principal de la aplicación, el cual comentaremos a continuación.

La sentencia:
Application.Initialize;

Invoca el apuntador al procedimiento initProc, el cual por defecto tiene el valor nil y salvo en circunstancias especiales no debería modificarse.

La sentencia:
Application.CreateForm (Tform1, Form1) ;
Crea la forma principal de la aplicación.

Finalmente la sentencia:
Application.Run;

Ejecuta la aplicación, desplegando la forma Form1; se puede agregar código al bloque para propósitos de inicialización especiales.

6.1.2 Archivos de Unidades

"Las unidades son archivos de código fuente con la extensión .pas. Existen tres tipos básicos de unidades.

- **Unidades de Formas/Data Modules.** Son unidades que se generan automáticamente por Delphi, existe una unidad por cada forma o data module que contenga un proyecto.

- **Unidades de Componentes.** Son unidades creadas por el desarrollador o por Delphi (a través del asistente de componentes) que contienen el código fuente de un componente.

- **Unidades de Propósito General**. Son unidades que contienen tipos de datos, declaraciones de clases, procedimientos y funciones, que están disponibles para utilizarse dentro de la aplicación."[1]

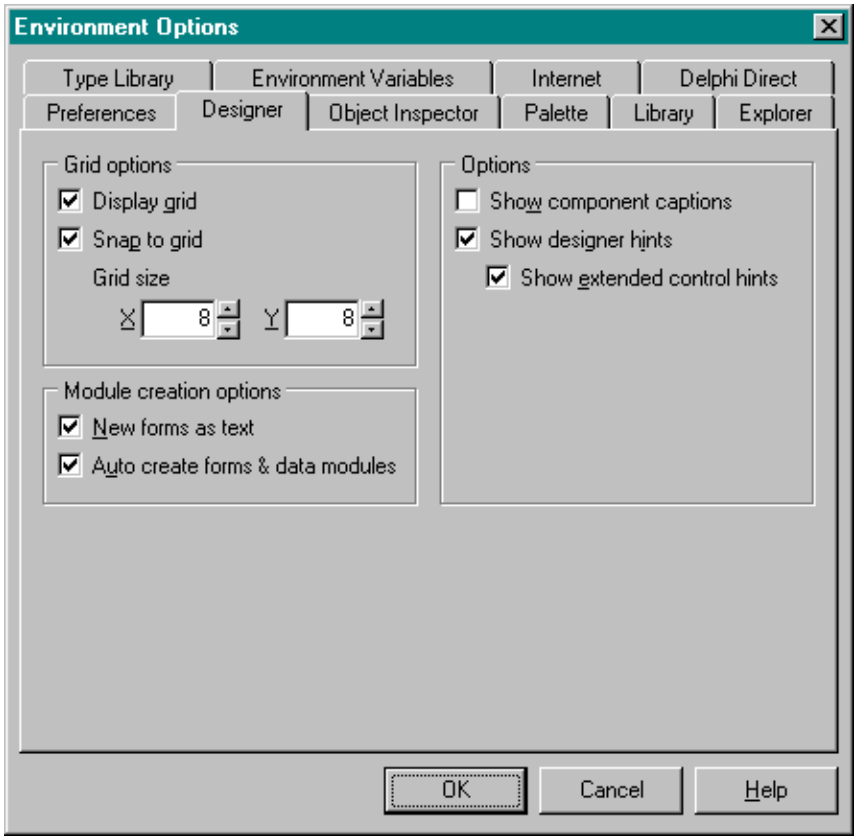
[1] Ayuda del Software Delphi 7

En el caso de un proyecto nuevo de Delphi, este solamente contiene una unidad Unit1, correspondiente a la forma: Form1, cuyo contenido se lista a continuación.

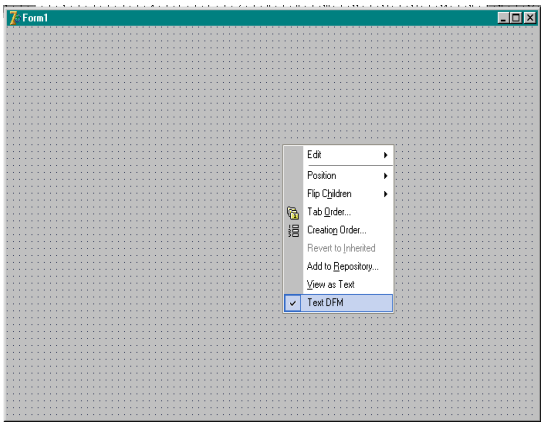
```
Unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs;
type
TForml = class (TForm)
Private
   (Private declarations)
public
   (Public declarations)
end;

var
   Form1: Tform1;
implementation
 {$R *.dfm}
end.
```

6.1.3 Archivos de Formas

Un archivo de forma es una representación de una forma, el cual puede salvarse en formato binario o texto, los archivos de forma tienen una extensión .dfm para proyectos de VCL o .xfm para proyectos de CLX, el formato recomendado es el de texto porque este puede ser manejado más fácilmente por sistemas de control de versiones. Para seleccionar el tipo de formato con el que se desea salvar los archivos de formas, seleccione del menú de Delphi Tools | Environment Options | Designer y marque la casilla de verificación New Forms as Text, tal como muestra en la siguiente la figura.



También es posible cambiar el tipo d formato con el que se guarda un archivo de forma pulsando click con el botón derecho del mouse sobre la forma en el diseñador de formas y seleccionar o no la opción Text DFM, tal como se muestra en la siguiente pantalla:

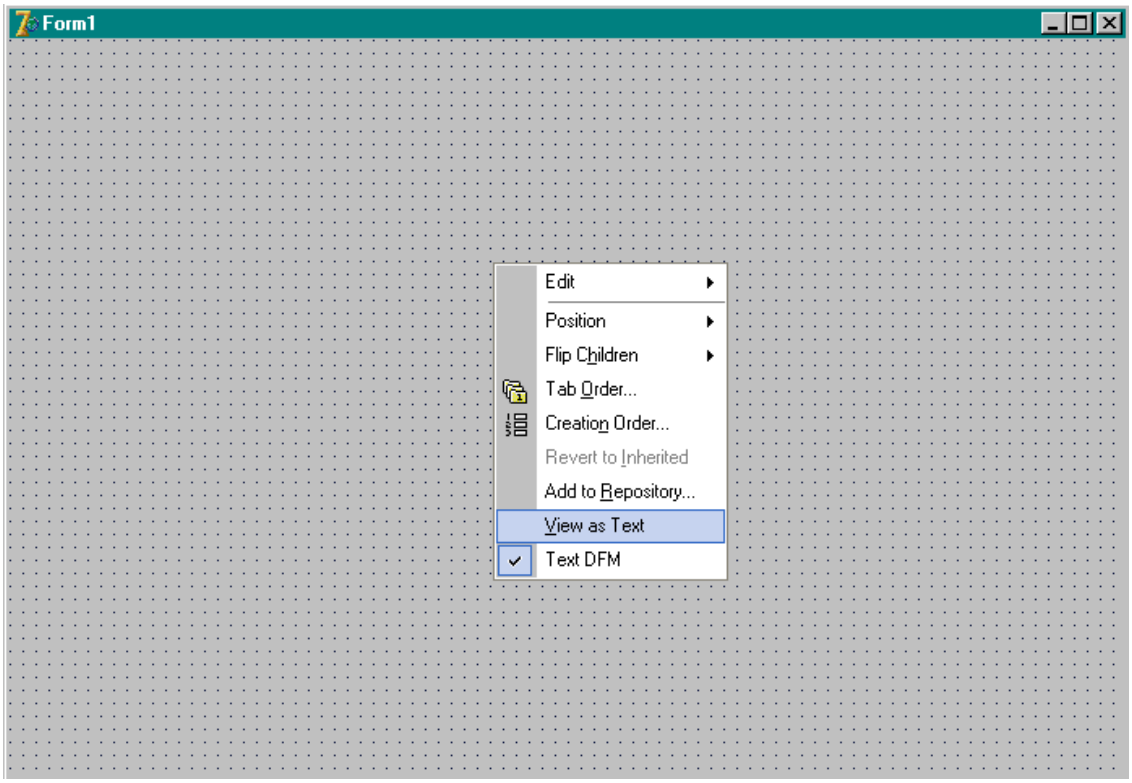


Siempre que se crea una nueva forma, Delphi crea una unidad con la extensión .pas y un archivo de forma con la extensión .dfm, las unidades de código relacionadas a las formas o a los módulos de datos contiene la siguiente línea de código.

*{$R *.dfm}*

Esto le indica al compilador que debe ligar el archivo correspondiente de la forma al proyecto, pues los archivos de forma son un tipo de archivos de recurso.

Para observar el contenido de un archivo de forma se debe pulsar click con el botón derecho del mouse sobre la forma y seleccionar la opción View as Text, o seleccionar la combinación de teclas Alt+F12.



Con este proceso se mostrará, en lugar de la forma, una unidad de código con un listado parecido al siguiente:

object Form1: TForm1
Left = 192
Top = 107
Width = 696
Height = 480
Caption = 'Form1'

```
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
end
```

Observar la representación textual del archivo de forma es útil porque nos muestra los valores de las propiedades de la forma y de los objetos que contiene que difieren de su valor por defecto.

Normalmente no se debería de editar el archivo de forma manual porque esto podría resultar en un error de lectura, lo cual impediría que Delphi pudiera abrir el archivo posteriormente.

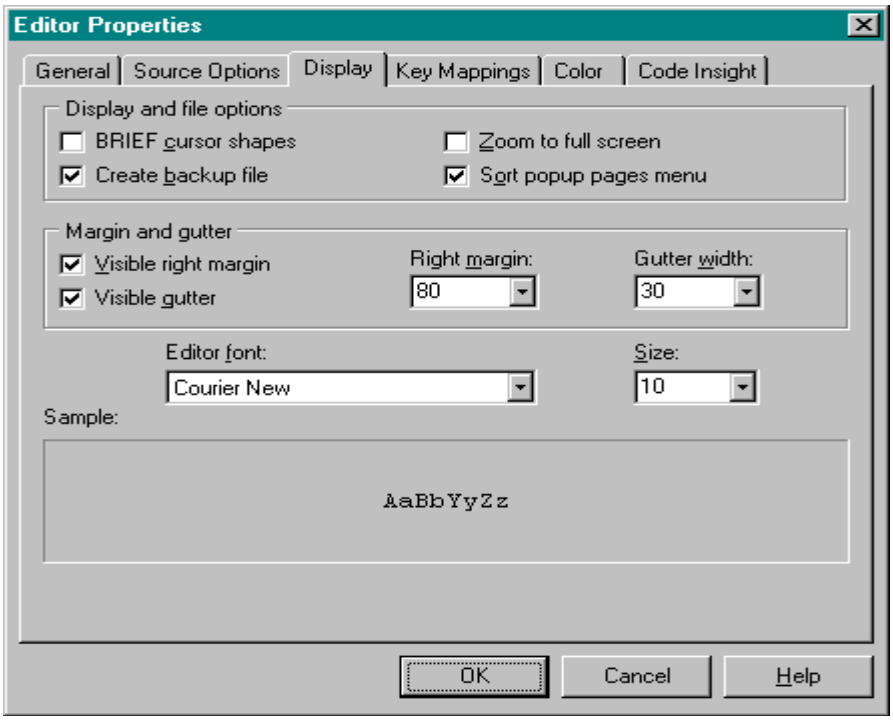
6.1.4 Archivos de Recursos

Los archivos de recursos contienen datos binarios llamados recursos, los cuales son ligados al archivo ejecutable de la aplicación. Delphi crea automáticamente un archivo con extensión .res, el cual contiene el icono del proyecto, y la información de versión del mismo, se pueden agregar recursos adicionales a un proyecto generando nuevos archivos de recursos y ligándolos al proyecto. Estos archivos de recursos se pueden crear con editores de recursos como el Editor de Imágenes de Delphi o la herramienta Resource Workshop.

6.1.5 Archivos de Respaldo

Delphi crea archivos de respaldo para el archivo del proyecto .dpr, para los archivos de unidades .pas y para los archivos de forma .dfm. Estos archivos de respaldo contienen la última copia del archivo antes de que se salvara por última vez el archivo. El archivo de respaldo del proyecto tiene la extensión .~dpr, los archivos de respaldo de unidades tienen la extensión .~pas, y los archivos de respaldo de los archivos de forma tienen la extensión .~dfm.

Se puede configurar la creación de los archivos de respaldo, a través del menú de Delphi Tools | Editor Options, que presenta un cuadro de diálogo en el cual debemos presentar la pestaña Display seleccionar o no la casilla de verificación Create backup file.



6.1.6 Archivos de Paquetes

Un paquete es una librería de ligado dinámico especial utilizada por las aplicaciones de Delphi, el IDE, o ambos. Los paquetes de tiempo de ejecución proporcionan funcionalidad cuando se ejecuta una aplicación. Los paquetes de tiempo de diseño se utilizan para instalar componentes en el IDE y para crear editores de propiedades que permiten personalizar a los componentes. Un solo paquete puede funcionar tanto en tiempo de diseño como en tiempo de ejecución, y los paquetes de tiempo de diseño frecuentemente trabajan haciendo llamadas a paquetes de tiempo de ejecución. Para distinguir a los paquetes de otras DLL's, las bibliotecas de paquetes se almacenan en archivos con la extensión .bpl.

Como otras librerías de rutinas, los paquetes contienen código que puede ser compartido entre distintas aplicaciones. Por ejemplo, los componentes de la VCL más frecuentemente utilizados residen en un paquete llamado vcl (visualclx en aplicaciones de CLX). Cada vez que se crea una nueva aplicación de VCL, esta automáticamente utiliza este paquete. Cuando se compila esta aplicación, la imagen ejecutable de la aplicación contiene solamente el código y datos únicos de la aplicación, el código en común esta en el paquete de tiempo de ejecución llamado vcl70.bpl. Una computadora con varias aplicaciones instaladas basadas en paquetes solamente necesita una copia de vcl70.bpl, la cual es compartida por todas las aplicaciones y por el IDE.

Varios paquetes de tiempo de ejecución encapsulan los componentes de VCL y CLX mientras varios paquetes de tiempo de ejecución manipulan componentes en el IDE.

Se pueden construir aplicaciones en Delphi con o sin paquetes. Sin embargo para agregar componentes al IDE, estos se deben instalar en paquetes en tiempo de diseño.

6.2 Tips de Manejo de Proyectos

Existen varias formas de optimizar el proceso de desarrollo utilizando técnicas que faciliten una mejor organización y reutilización de código, a continuación presentaremos algunos tips para aplicar dichas técnicas.

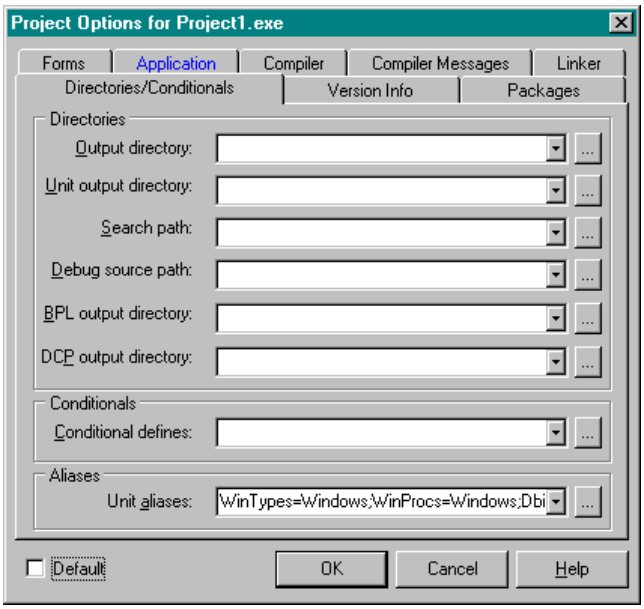
Un Proyecto, Un Directorio

En general es una buena idea administrar los proyectos de tal forma que los archivos de un proyecto estén separados de los archivos de otros proyectos.

Unidades para Compartir Código

Se pueden compartir rutinas utilizadas comúnmente por otras aplicaciones colocando dichas rutinas en unidades que puedan ser accedidas por múltiples proyectos. Típicamente, se puede crear un directorio de utilidades en disco y colocar las unidades en dicho directorio. Cuando se necesite acceder una función particular que exista en una de las unidades de dicho directorio, solamente se coloca el nombre de la unidad en la cláusula uses de la unidad o proyecto que requiera acceso al archivo.

Además se debe agregar la ruta del directorio de utilidades a la ruta de búsqueda (Search Path) en la pestaña Directories/Conditionals del cuadro de diálogo de Opciones del Proyecto, al cual se accede a través del menú de Delphi: Project │ Options, el cual se muestra a continuación.



Si se utiliza el Administrador de Proyectos, se puede agregar una unidad de otro directorio a un proyecto existente, de esta manera se agrega el directorio automáticamente a la ruta de búsqueda.

Hacer que una Forma utilice otras formas

Solo porque cada forma está contenida en su propia unidad no significa que no pueda acceder a las variables, propiedades y métodos de otras formas. Delphi genera código en el archivo .pas del proyecto, declarando la instancia de la forma como una variable global. Todo lo que se necesita es agregar el nombre de la unidad definiendo una forma particular a la cláusula uses de la unidad que define la forma que necesita acceso.

Por ejemplo, si la forma Form1 definida en la unidad Unit1.pas necesita acceso a la forma Form2, definida en la unidad Unit2.pas, sola basta con agregar Unit2 a la cláusula uses de Unit1.

unit Unitl
interface
...
implementation
Unit2;
...
end;

Ahora Unitl puede hacer referencia a Form2 en cualquier parte de su sección implementation.

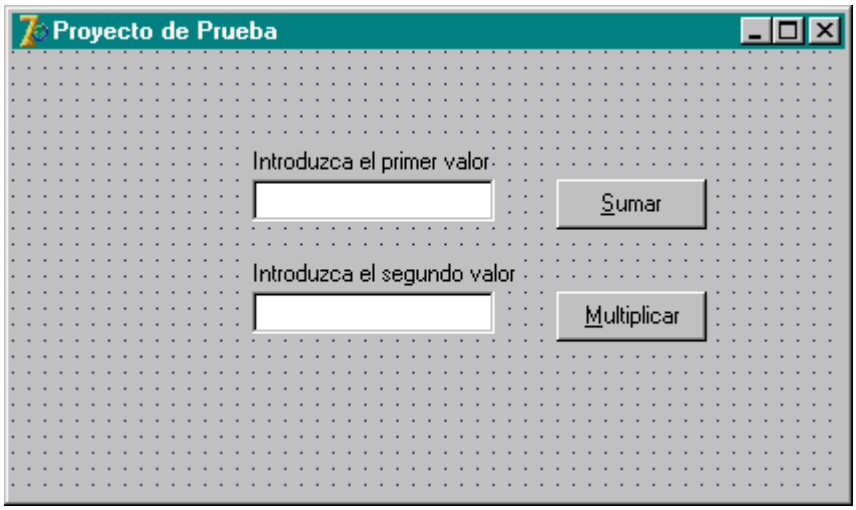
Aplicación de Ejemplo

"Construiremos una aplicación para mostrar los diferentes tipos de archivos de un proyecto en Delphi y su interacción.

1. Cree un proyecto nuevo, a través del menú de Delphi7 File | New | Application.
2. Agregue en su forma dos componentes TLabel, dos componentes TEdit y dos componentes TButton desde la pestaña Standard de la Paleta de componentes, modifique las propiedades de dichos componentes tal como se muestra en la tabla siguiente.

COMPONENTE	PROPIEDAD	VALOR
Labe1	Caption	Introduzca el primer valor
Label2	Caption	Introduzca el segundo valor
Edit1	Text	
Edit2	Text	
Button1	Caption Name	&Sumar btnSumar
Button2	Caption Name	&Multiplicar btnMultiplIcar
Form1	Caption	Proyecto de Prueba

3. Forma del proyecto debería verse como la siguiente figura:



4. A continuación salve su proyecto, seleccionando del menú de Delphi la opción: File | Save All, guarde la Unidad Unit1 con el nombre Umain.pas y el archivo del proyecto con el nombre ProyectoEjemplo.dpr.

5. A continuación agregamos una unidad de propósito general a nuestro proyecto, para esto, seleccione del menú de Delphi la opción: File | New | Unit, se creará una segunda unidad en el proyecto.

6. Modifique el código de la unidad tal como se muestra en el siguiente listado:

```
unit Ucalculos;

interface
function Suma(num1:integer; num2:integer):integer;
function Multiplica(num1:integer; num2:integer):integer;
implementation
function suma(num1:integer; num2:integer):integer;
begin
  result := num1 + num2;
end;
function Multiplica(num1:integer; num2:integer):integer;
begin
   result := num1 * num2;
end;
end.
```

7. Grabe su nueva unidad con el nombre Ucalculos.pas
8. Localice la sección Implementation de la unidad de código UMain y escriba la siguiente línea de código:

```
Uses Ucalculos;
```

El codigo anterior permite utilizar las funciones Suma y Multiplica declaradas en la unidad Ucalculos.pas desde la unidad Umain.pas

9. Seleccione el botón btnSumar y localice dentro del Object Inspector la pestaña Events, recuerde que ésta muestra la lista de eventos definidos para el botón.
10. Seleccione el evento OnClick del botón y pulse doble click sobre la sección en blanco que se muestra a la derecha de éste, Delphi generará automáticamente un manejador de evento en la Unidad de Código Umain.
11. Escriba el siguiente código:

```
procedure Tform1 .btnSumarCIick (Sender; TObject);
var
num1, num2 : Integer ;
begin
   num1 := StrToInt(Edit1.Text);
   num2 := StrToInt(Edit2.Text) ;
   ShowMessage(IntToStr(Suma(num1, num2) ) ) ;
end;.
```

12. Seleccione el botón btnMultiplicar y en el Object Inspector pulse doble click sobre la sección en blanco de! evento OnClick para escribir el siguiente código en el manejador del evento:

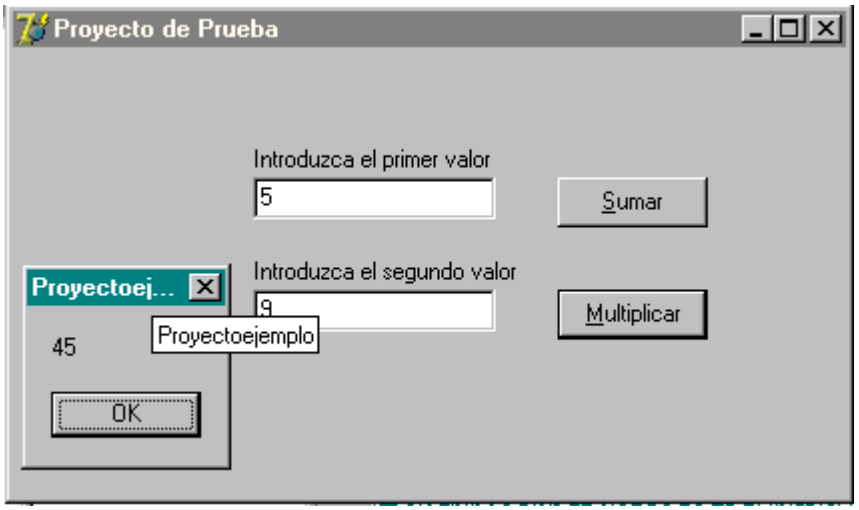
```
procedure Tform1. btnMultiplicarClick (Sender : TObject) ;
var
num1, num2: Integer ;
begin
   num1 := StrToInt(Editl.Text);
   num2 := StrToInt(Edit2.Text);
   ShowMessage(IntToStr(Multiplica(numl, num2))) ,-
end;
```

Su unidad de código deberá tener el siguiente aspecto:

```
unit Umain;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    btnsumar: TButton;
    btnmultiplicar: TButton;
```

```
    procedure btnsumarClick(Sender: TObject);
    procedure btnmultiplicarClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

implementation

{$R *.dfm}
uses Ucalculos;
procedure TForm1.btnsumarClick(Sender: TObject);
var
 num1, num2: integer;
begin
 num1 := strtoint(edit1.Text);
 num2 := strtoint(edit2.Text);
 showmessage(inttostr(suma(num1, num2)));
end;
procedure TForm1.btnmultiplicarClick(Sender: TObject);
var
 num1, num2: integer;
begin
  num1 := strtoint(edit1.Text);
  num2 := strtoint(edit2.Text);
  showmessage(inttostr(Multiplica(num1, num2)));
end;
end.
```

13. Ejecute y pruebe su proyecto."[2]



[2] http://www.terra.es/personal/resfer/delphi/

6.3 Preguntas y Ejercicio de Repaso

1. Mencione los tipos de archivos que forman un proyecto en Delphi.

2. Subraya que extencion tiene el proyecto principal.

a).pas b).dcu c).dpr d).ddp

3. Mencione las formas en que se pueden almacenar los archivos de formas.

4. Define los tres tipos básicos de unidades.

5. ¿Cómo se puede agregar un recurso a un proyecto?

6. Describa la funcionalidad de los paquetes.

7. Crea un programa que obtenga el factorial de un número.

8. ¿Los archivos de respaldo que extención tienen?

a).pas b)~.pas c).dpr d).dcu

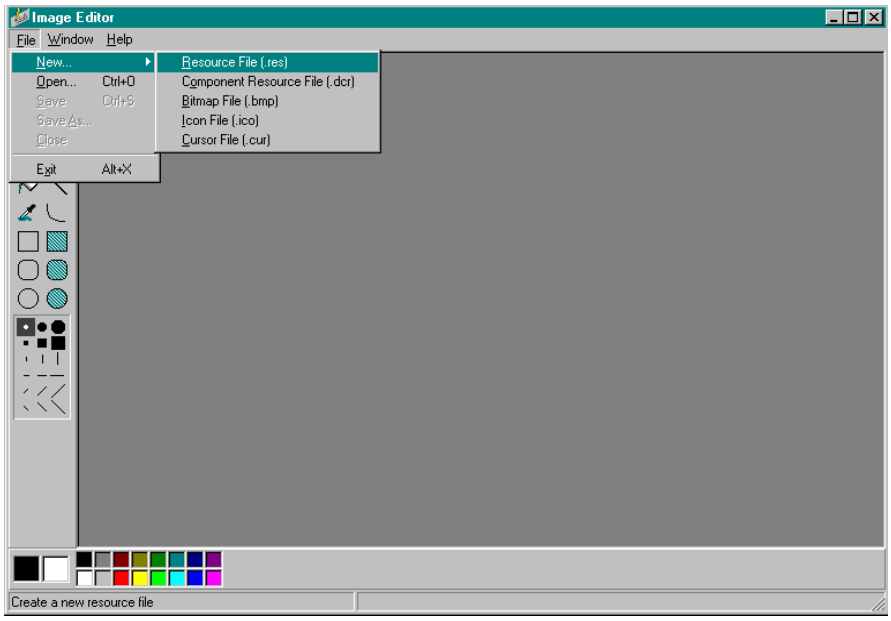
9. Crea tu propio icono y pruebalo en cualquier aplicación de Delphi.

Ejercicio de Repaso

Construcción de un Archivo de Recursos.

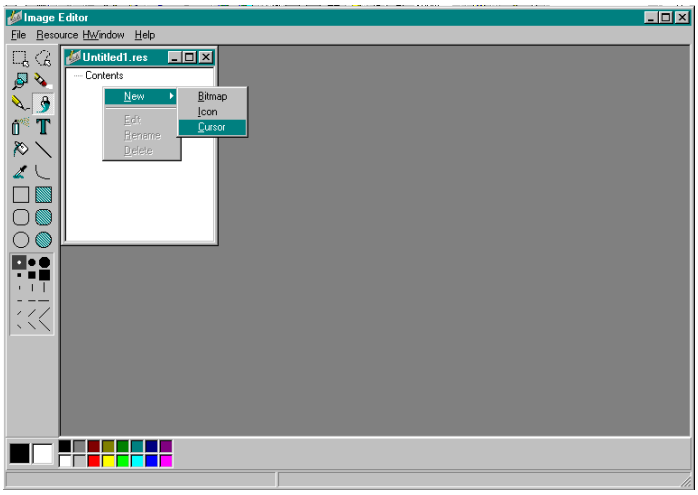
A continuación se especifican los pasos a realizar para generar un archivo de recursos para almacenar un cursor para la aplicación que desarrollamos en este capitulo.

1. Del menú de Delphi seleccione la opción Tools | Image Editor, para invocar al Editor de Imágenes.
2. Dentro del editor de imágenes seleccione la opción File | New | Resource File (.res).

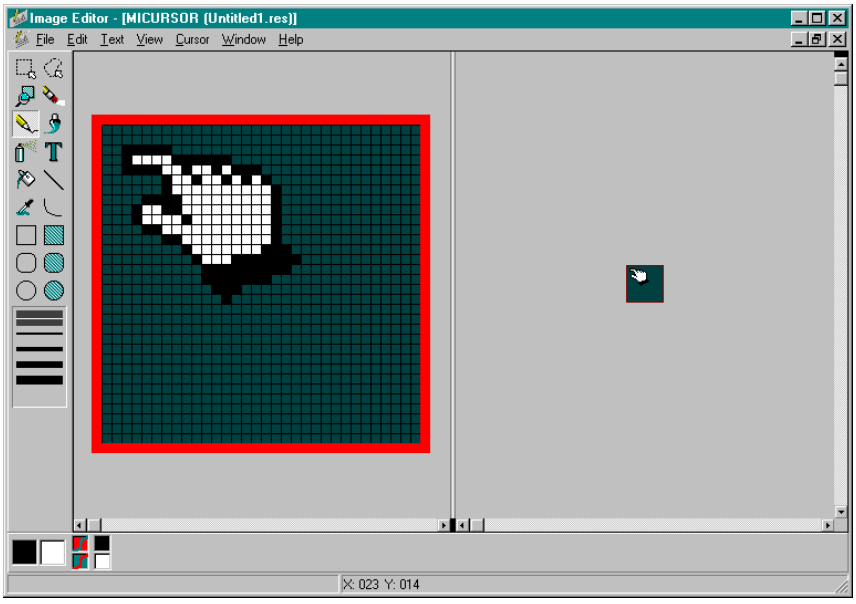


3. Salve el archivo de recursos en la misma carpeta que salvó el proyecto con el nombre Recursos.res.

4. Dentro de la ventana del recurso, pulse click con el botón derecho sobre la palabra Contents, y seleccione la opción New | Cursor, tal como se muestra en la siguiente figura.



5. Dentro de la ventana del recurso, pulse click derecho sobre el elemento Cursor1 y seleccione la opción Rename, cambie del elemento a MICURSOR.
6. Pulse doble click sobre el elemento MICURSOR, esto nos permite editar la imagen del cursor, modifique el curso tal como se muestra en la siguiente pantalla.



7. Salve las modificaciones y cierre el Editor de Imágenes,
8. A continuación le indicaremos a nuestro proyecto que cargue el nuevo cursor. Para esto, seleccione el archivo del proyecto a través del menú de Delphi Project | View Source.
9. Modifique el archivo tal como se muestra en el siguiente listado.

```
program P_ProyectoEjemplo;
uses
```

```
Forms, Windows,
U_FrPrincipal in 'U_FrPrincipal.pas' {Form1),
{$R *.res)
{$R Recursos.res}
const crMyCursor = 5;
begin
   Application.Initialize;
   Application.CreateForm(TForml, Form1);
   Screen.Cursors[crMyCursor] := LoadCursor(HInstance, 'MICURSOR');
   Screen.Cursor := crMyCursor;
   Application.Run;
end.
```

Para cargar el cursor desde el archivo de recursos Recursos.res, primero debemos indicarle a Delphi que ligue el archivo al proyecto, esto se realiza a través de la línea.

{$R Recursos.res}

El objeto Screen posee un arreglo de cursores al cual debemos agregar nuestro cursor personalizado, esto lo logramos declarando la constante:

const crMyCursor = 5;

A continuación obtenemos el cursor desde el archivo de recursos y lo agregamos al arreglo de cursores del objeto Screen, por medio de las lineas:

Screen.Cursors[crMyCursor] := LoadCursor(HInstance, 'MICUPSOR') ;

Screen.Cursor := crMyCursor;

La función LoadCursor se encuentra definida en la unidad Windows.pas, por lo cual debemos agregar a la cláusula uses del proyecto dicha unidad.

10. Ejecute su proyecto (F9), observe como el cursor de la aplicación se carga desde el archivo de recursos.

Por último, no edite el archivo de recursos que Delphi crea automáticamente para el proyecto en tiempo de compilación, debido a que cualquier modificación se perderá cuando se vuelva a compilar el proyecto. Recuerde que si desea agregar recursos adicionales a un proyecto debe crear archivos de recursos separados, con un nombre distinto al del proyecto y agregar al archivo del proyecto la directiva ($R MIARCHIVO.RES}.

VII. Desarrollo de Aplicaciones Win32

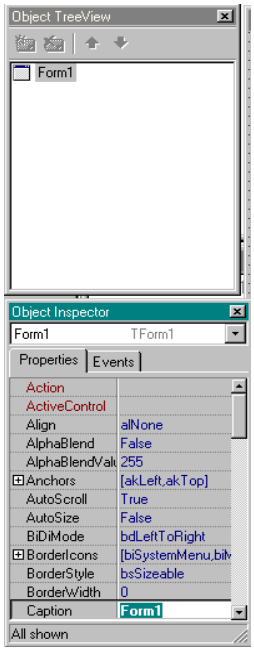
7.1 Características de las Aplicaciones Win32.

Delphi 7 permite el desarrollo de aplicaciones Win32 con diversas características, a continuación presentaremos ejemplos de distintas características típicas de las aplicaciones Win32 y su implementación en Delphi.

7.1.1Drag and Drop

"Este proceso define cómo objetos que se encuentran en una aplicación o ventana determinada pueden ser arrastrados manteniendo presionado el botón del mouse y soltados en otra aplicación o ventana, al dejar de apretar el botón correspondiente. La mayoría de los sistemas operativos gráficos, como Windows utilizan Drag and Drop.

Dentro de Delphi, percibimos el uso común de esta técnica cuando trabajamos en nuestras aplicaciones arrastrando y soltando componentes dentro de las formas. Es posible combinar la técnica Drag and Drop con otra característica de Windows que es Doking. El efecto Docking consiste en arrastrar un objeto e incorporarlo dentro del área de otro, tal como ocurre dentro del IDE de Delphi con las ventanas del Object TreeView y el Object Inspector"[1], por ejemplo.



[1] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

Aplicación de Ejemplo

1. Cree una aplicación nueva por medio del menú: File | New | Application.
2. Agregue desde la paleta de componentes, dos componentes TEdit y dos componentes TLabel desde la pestaña Standard de la Paleta de Componentes, modifique las propiedades de los componentes como se indica en la siguiente tabla:

COMPONENTE	PROPIEDAD	VALOR
Label1	Caption	Origen
Label2	Caption	Destino
Edit1	Text Name	SourceEdit
Edit2	Text Name	DestinationEdit

El componente SourceEdit servirá como fuente para el texto que se arrastrará y se soltará en el componente DestinationEdit para incorporar el texto dentro de él.

3. Seleccione el componente SourceEdit y desde el Object Inspector modifique su propiedad DragMode al valor dmAutomatic.

La propiedad DragMode determina la manera en que se inicia el arrastre de un objeto mediante dos valores diferentes:

- **DmAutomatic** La función de arrastre comienza automáticamente al momento que el usuario presiona el botón izquierdo del mouse sobre el componente. Esta opción tiene la ventaja de que no es necesario escribir código para controlar la función Drag and Drop..

- **DmManual** Permite iniciar la acción de arrastre mediante el uso del evento BeginDrag y los eventos del mouse.

4. A continuación, seleccione el componente DestinationEdit y genere un manejador de evento para el evento OnDragOver, pulsando doble click en la sección en blanco del evento desde el Inspector de Objetos, y modifique el código del manejador tal como se muestra en el siguiente listado.

```
procedure TForm1.DestinationEditDragOver(Sender, Source: TObject; X,
  Y: Integer; State: TDragState; var Accept: Boolean);
begin
```

```
  Accept := True;
end;
```

El evento OnDragOver ocurre cuando el usuario arrastra un objeto sobre un control, este evento define la variable Accept la cual permite indicar si el componente puede aceptar un objeto que ha sido arrastrado o no.

5. Genere un manejador de evento para el evento OnDragDrop, del componente DestinationEdit, pulsando doble click en la sección en blanco del evento desde el Inspector de Objetos, y modifique el código del manejador tal como se muestra en el siguiente listado

```
procedure TForm1.DestinationEditDragDrop(Sender, Source: TObject; X,
  Y: Integer);
begin
  destinationedit.Text := SourceEdit.Text;
end;
```

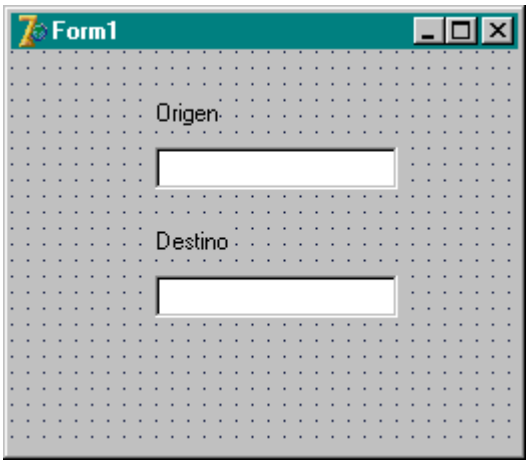
El evento OnDragDrop ocurre cuando el usuario suelta o libera un objeto que previamente fue arrastrado sobre otro objeto.

6. Salve su aplicación con los siguientes nombres:

Unidad de la forma: UMain.pas

Archivo del proyecto: DragandDrop.dpr

Su forma deberá lucir como la siguiente figura.



7. Ejecute su aplicación (F9), escriba algún texto en el componente SourceEdit y pulse click sobre él, sin soltar el botón del mouse, arrastre el texto sobre el componente DestinationEdit. Cuando suelte el botón del mouse el texto del componente DestinationEdit cambiará a lo que estaba escrito en el SourceEdit.

7.1.2 Eventos de Teclado

Una de las actividades comunes al desarrollar aplicaciones Win32 es el manejo de eventos de teclado, por ejemplo, es bastante común tener que capturar datos del usuario en componentes TEdit que solo acepten caracteres alfabéticos, o numéricos.

Para ejemplificar lo anterior realizaremos un a nueva aplicación.

"Aplicación de Ejemplo

1. Cree una aplicación nueva por medio del menú: File | New | Application.
2. Agregue desde la paleta de componentes, dos componentes TEdit y dos componentes TLabel desde la pestaña Standard de la Paleta de Componentes, modifique las propiedades de los componentes como se indica en la siguiente tabla:

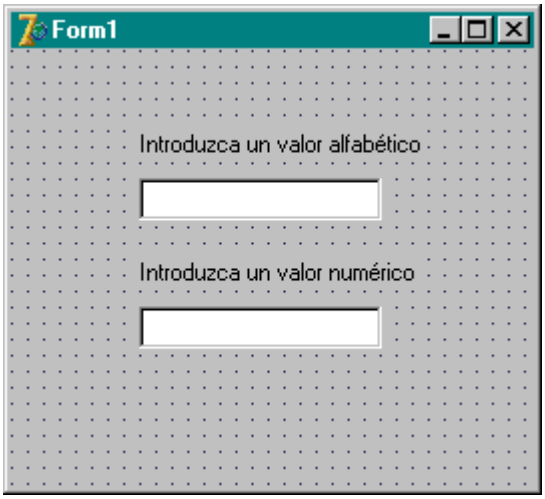
COMPONENTE	PROPIEDAD	VALOR
Label1	Caption	Introduzca un valor alfabético:
Label2	Caption	Introduzca un valor numérico:
Edit1	Text Ñame	EdAlfabetico
Edit2	Text Name	EdNumerico

3. Salve su aplicación con los siguientes nombres:

Unidad de la forma: UMain.pas

Archivo del proyecto: EventosTeclado.dpr

Su forma deberá lucir como la siguiente figura.



4. A continuación genere un manejador de evento para el evento OnKeyPress del componente EdAlfabetico, pulsando doble click en la sección en blanco del evento desde el Inspector de Objetos, y modifique el código del manejador tal como se muestra en el siguiente listado.

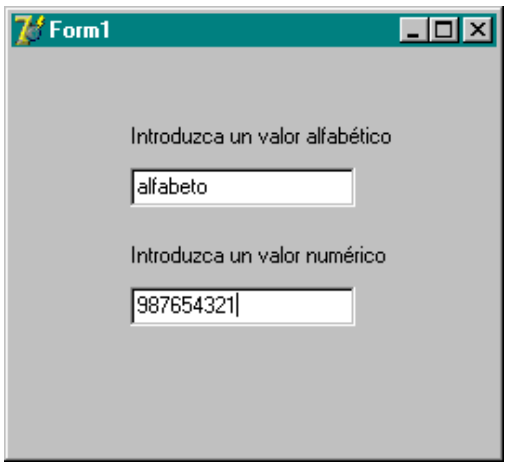
```
procedure TForm1.edalfabeticoKeyPress(Sender: TObject; var Key: Char);
begin
{#8 BkSp
#l3 Enter
#18 Tab}
if not (key in ['a'..'z', 'A'..'Z', #8, #13, #18]) then
   Key := Char(0);
end;
```

5. Seleccione el componente EdNumerico y genere un manejador de evento para el evento OnKeyPress; modifique el código del manejador como se muestra en el siguiente listado:

```
procedure TForm1.ednumericoKeyPress(Sender: TObject; var Key: Char);
begin
{#8 BhSp
#13 Enter
#18 Tab}
if not (key in ['0'..'9', #8, #13, #18] ) then
```

```
  Key := Char(0);
end;
```

6. Ejecute la aplicación, pruebe escribir información en los Edits, verifique que el componente EdAlfabetico acepta solamente caracteres alfabéticos y el componente EdNumerico acepta solamente caracteres numéricos."[2]



7.1.3 XP Themes

Al desarrollar aplicaciones de Win32 y ejecutarlas en Windows XP, las aplicaciones desarrolladas en Delphi7 normalmente no adoptan la apariencia estándar de Windows XP, esto se debe a Microsoft ha dividido los controles visuales comunes en dos versiones separadas. La versión 5 está disponible, en las versiones de Windows desde Windows 95 o superior, y despliega los controles con una apariencia 3D. La versión 6 está disponible en Windows XP, en esta versión los controles son visualizados a través de un motor de temas el cual muestra el tema actual en Windows XP, si el usuario modifica el tema, los controles comunes de la versión 6 mostrarán la nueva versión automáticamente, no es necesario recompilar la aplicación.

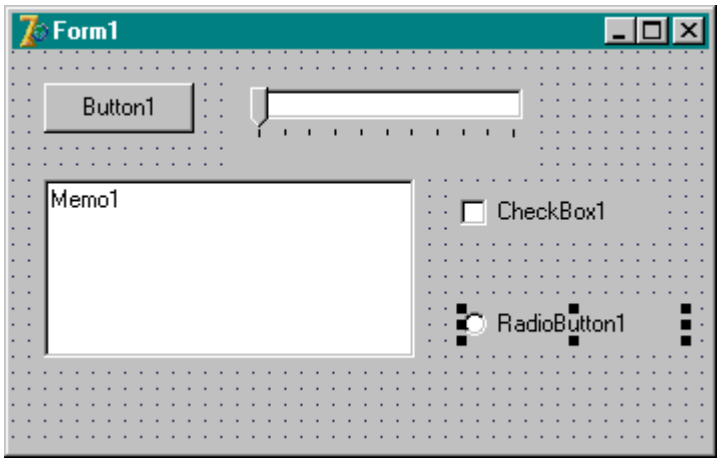
La VCL puede adaptarse a ambos tipos de controles comunes. Borland ha agregado un número de componentes a la VCL para manejar los controles comunes de manera automática y transparente. Por defecto, todas las aplicaciones de VCL desplegarán los controles comunes de la versión 5. Para desplegar los controles de la versión 6, se debe agregar un componente TXPManifest desde la paleta Win32 de la Paleta de Componentes.

Para comprobar el funcionamiento del componente TXPManifest es necesario desarrollar el siguiente ejercicio en Windows XP.

[2] http://www.clubdelphi.com/

Aplicación de Ejemplo

1. Cree una aplicación nueva por medio del menú: File | New | Application.
2. En su forma, agregue un componente TButton, un componente TMemo, un componente TCheckBox, un componente TRadioButton, desde la pestaña Standard de la Paleta de Componentes y un componente TTrackBar de la pestaña Win32, de tal manera que su forma luzca como la siguiente figura:



3. Ejecute la aplicación.

Note como la aplicación utiliza el estilo visual de la versión 5 de los controles comunes, es decir no adopta el tema visual de Windows XP.

4. Agregue a la forma un componente TXPManifest desde la pestaña Win32 de la Paleta de Componentes y ejecute nuevamente la aplicación.

Note como al agregar el componente TXPManifest la aplicación utiliza el estilo visual de la versión 6 de los controles comunes, adoptando el tema visual de Windows XP.

7.2 Creación de un Editor de Texto

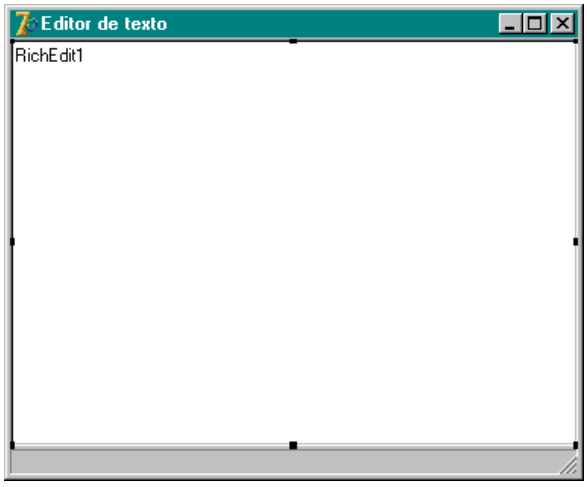
1. Cree una aplicación nueva por medio del menú: File | New | Application.
2. Salve la aplicación con los siguientes nombres de archivo:

Unidad de la forma: UMain.pas

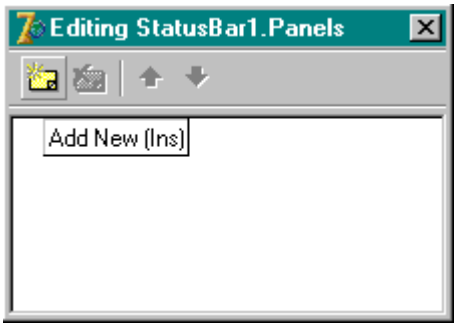
Archivo del proyecto: TextEditor.dpr

3. Modifique la propiedad Caption de la forma con el valor Editor de Texto.

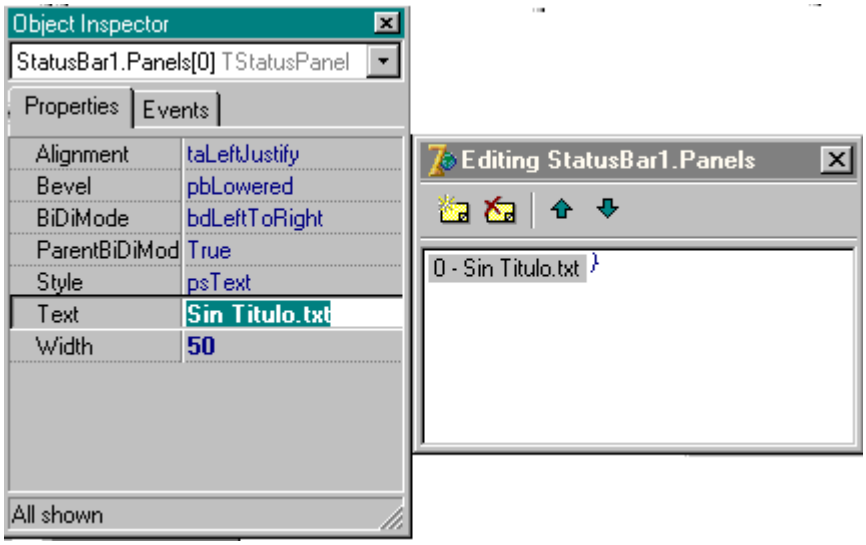
4. Agregue un componente TStatusBar (pestaña Win32) a la forma, pulsando doble click sobre él en la Paleta de Componentes.
5. Coloque un componente TRichEdit en la forma, desde la pestaña Win32 de la Paleta de Componentes, modifique su propiedad Align al valor alClient para que ocupe toda e! área libre de la forma. Su pantalla deberá tener el siguiente aspecto.



6. Pulse doble click en la barra de status para abrir el cuadro de diálogo StatusBar1.Panels.
7. Pulse click en el botón Add New para agregar un panel a la barra de status como se muestra en la siguiente figura.



8. Modifique la propiedad Text del nuevo panel agregado a la barra de status con el valor SinTitulo.txt. Con esto indicamos que cuando utilicemos el editor de textos si el archivo actualmente editado aun no ha sido salvado, el nombre de archivo será SinTitulo.txt.



9. Cierre el cuadro de dialogo StatusBar1.Panels.

Agregando soporte para un menú y una barra de herramientas

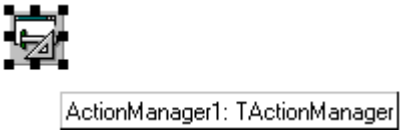
Nuestra aplicación de editor de texto necesita un conjunto de comandos que nos permitirán construir un menú y una barra de herramientas. Delphi proporciona los componentes TImageList y TActionManager que permiten centralizar las imágenes y las acciones que se incluirán en los menús y en las barras de herramientas.

Componente TImageList

El componente TImageList se encuentra en la pestaña Win32 de la paleta de componentes y es un componente no visual que permite almacenar una colección de imágenes clasificadas, cada una de las cuales puede recuperarse en la aplicación a través de su índice.

Componente TActionManager

El componente TActionManager se encuentra en la pestaña Additional de la paleta de componentes y es un componente no visual que proporciona un mecanismo para crear, configurar, desplegar y administrar todas las acciones (tareas, funciones, procedimientos, métodos, etc.) que utiliza y ejecuta una aplicación.



El componente TActionManager permite configurar y crear cualquier acción definida por el programador o trabajar con un conjunto de acciones estándar que han sido incorporadas previamente por Delphi y que definen las tareas más comunes de Edición, Formato, Navegación en DataSets, etc. La particularidad de estas acciones es que son códigos comunes en dos, tres o más objetos sobre los cuales aplicaremos la Reutilización de Código; una vez creadas estas acciones se ligan los objetos a la acción requerida por medio de la propiedad Action.

10. De la pestaña Win32 de la Paleta de Componentes, agregue un componente ImageList a la forma. Recuerde que el componente ImageList es un componente no visual, así que se puede colocar en cualquier parte de la forma.

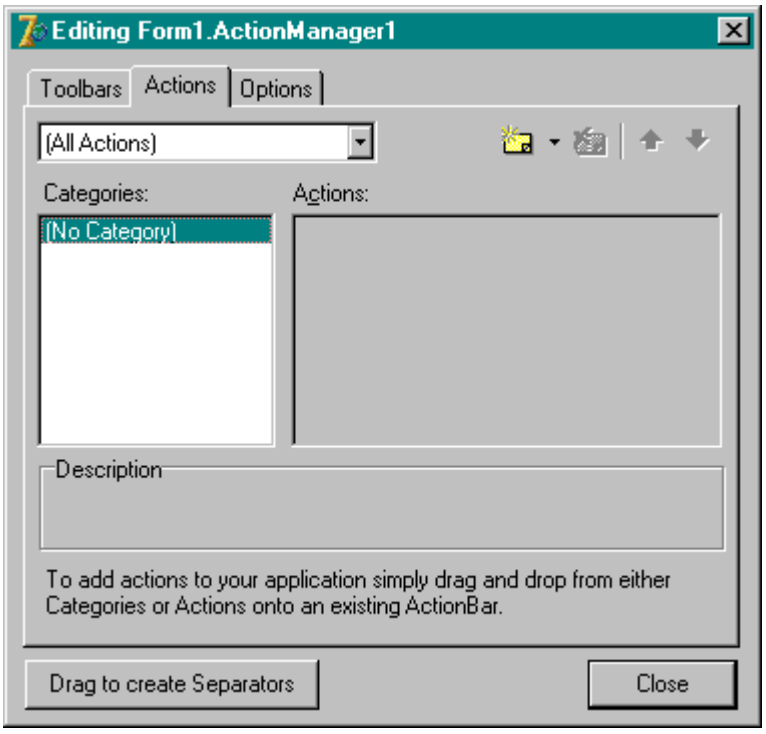
11. De la pestaña Additional agregue un componente ActionManager.

12. Modifique la propiedad Images del componente ActionManager1 a ImageList1.

Ahora agregaremos, las acciones necesarias al componente ActionManager1 para brindar funcionalidad a nuestra aplicación. Agregaremos acciones no estándar, para las cuales modificaremos todas sus propiedades e implementaremos el código necesario, y acciones estándar para .las .cuales las propiedades se fijaran automáticamente.

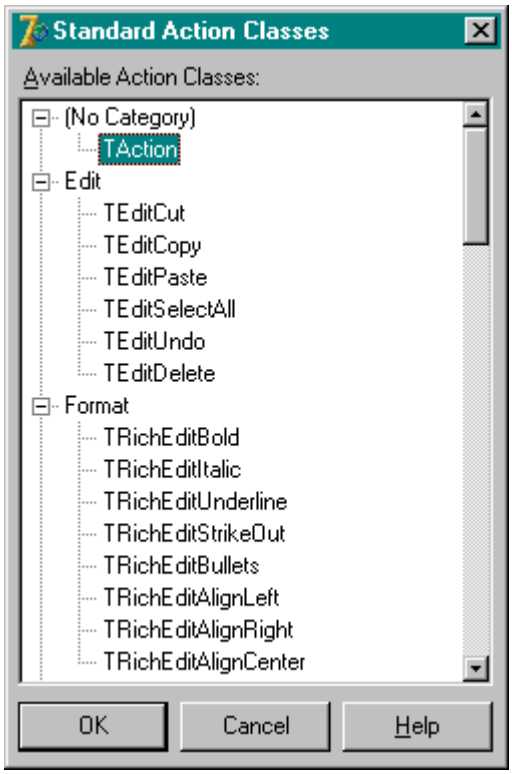
Agregando acciones estándar.

13. Pulse doble click sobre el componente ActionManagerl. Aparecerá el cuadro de diálogo de edición:

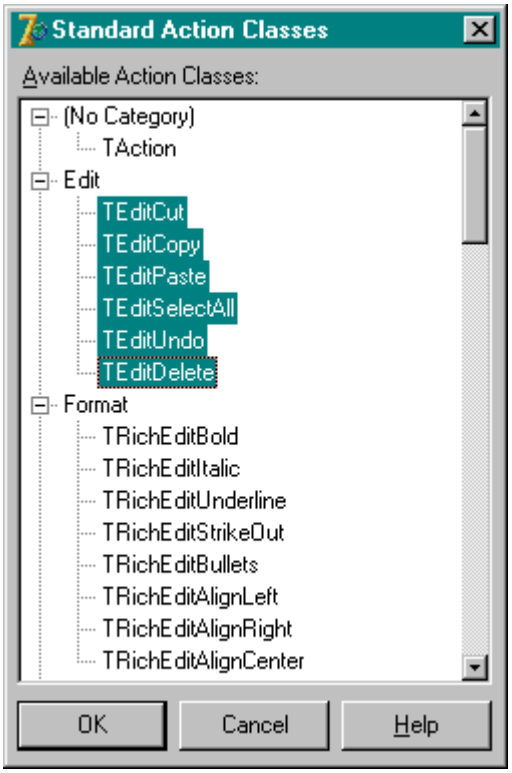


14. Dentro de este cuadro de dialogo, seleccione la pestaña Actions, pulse click en la lista desplegable cerca del botón New Action y seleccione la opción New Standar Action…

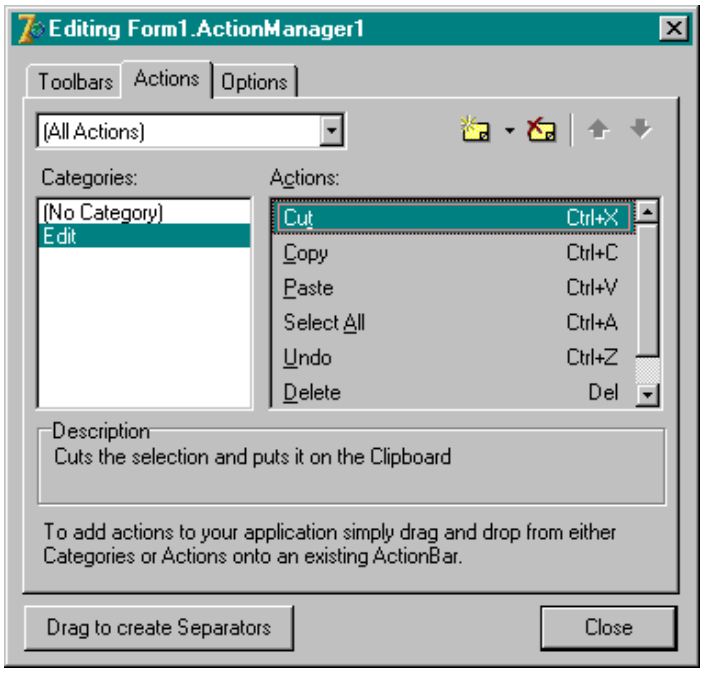
El cuadro de dialogo de clases de Acciones Estándar aparecerá como se muestra en la siguiente figura:



15. Dentro del cuadro de diálogo de Clases de Acciones Estándar ubique a la categoría Edit y presione la tecla Ctrl para seleccionar todas las acciones de esta categoría.



16. Pulse click en el botón OK para agregar las acciones seleccionadas en la lista de acciones del editor del Administrador de Acciones.

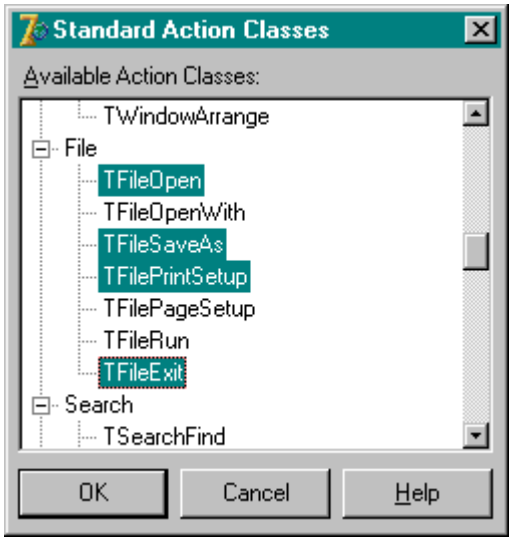


17. Seleccione cada una de estas acciones y modifique sus propiedades Caption con una descripción en español. Por ejemplo:

ACCION	PROPIEDAD	VALOR
EditCut1	Caption	&Cortar
EditCopyl	Caption	C&opiar
EditPastel	Caption	&Pegar
EditSelectAll1	Caption	&Seleccionar Todo
EditUndol	Caption	&Deshacer
EditDeletel	Caption	&Borrar

18. Pulse nuevamente click sobre la flecha desplegable cerca del botón New Action, y seleccione la opción New Standard Action.

19. Dentro del cuadro de diálogo de Clases de Acciones Estándar ubique la categoría File y seleccione las acciones: TFileOpen, TFileSaveAs, TFilePrintSetup y TfileExit.

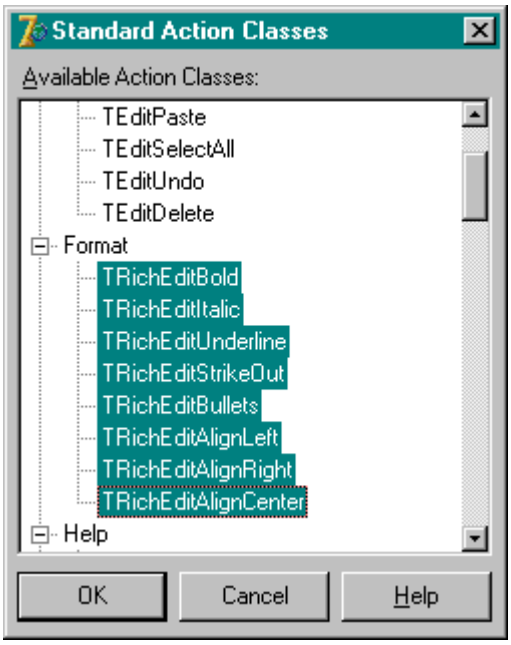


20. Pulse click en el botón OK para agregar las acciones seleccionadas en la lista de acciones del editor del Administrador de Acciones dentro de la categoría File.
21. Seleccione cada una de las nuevas acciones y modifique sus propiedades Caption con una descripción en español como anteriormente lo hicimos. Por ejemplo:

ACCION	PROPIEDAD	VALOR
FileOpen1	Caption	&Abrir…
FileSaveAs1	Caption	Salvar &Como…
FilePrintSetup1	Caption	Im&primir
FileExit	Caption	&Salir

22. Pulse nuevamente click sobre la flecha desplegable cerca del botón New Action, y seleccione la opción New Standard Action.

23. Dentro del cuadro de diálogo de Clases de Acciones Estándar ubique la categoría Format y seleccione todas las acciones de esta categoría.



24. Pulse click en el botón OK para agregar las acciones seleccionadas en la lista de acciones del editor del Administrador de Acciones dentro de la categoría Format.
25. Seleccione cada una de las nuevas acciones y modifique sus propiedades Caption con una descripción en español. Por ejemplo:

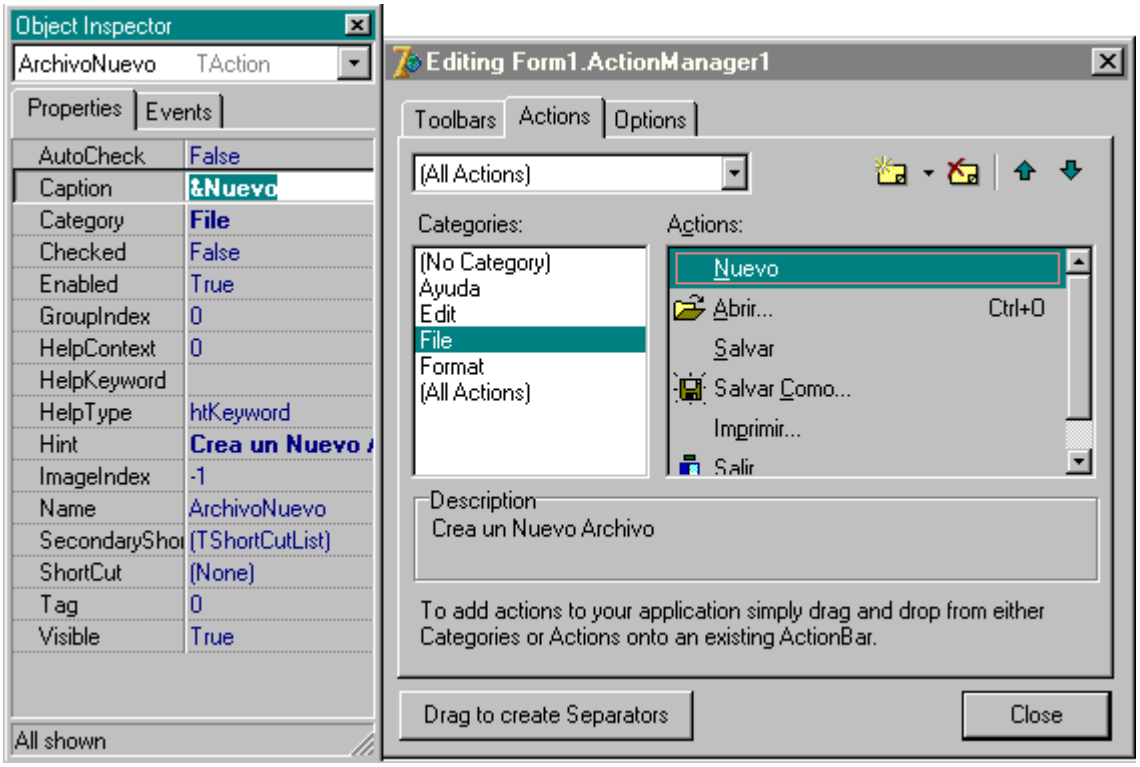
ACCION	PROPIEDAD	VALOR
RichEditBoldl1	Caption	&Negrita
RichEditItalic1	Caption	&Cursiva
RichEditUnderlinel	Caption	&Subrayado
RichEditStrikeOut1	Caption	&Tachado
RichEditBulletsl	Caption	&Viñetas
RichEditAlignLeft1	Caption	Alineación l&zquierda
RichEditAlignRight1	Caption	Alineación &Derecha
RichEditAlignCenter1	Caption	Cent&rado

Agregando acciones definidas por el usuario

26. Dentro del editor del Administrador de Acciones seleccione la categoría File y pulse nuevamente click sobre la flecha desplegable cerca del botón New Action y ahora seleccione la opción New Action.

27. En el Inspector de Objetos modifique las propiedades de la nueva acción como se indica en la siguiente tabla.

PROPIEDAD	VALOR
Caption	& Nuevo
Hint	Crea un nuevo archivo
Name	Archivo Nuevo



28. Agregue una nueva acción más, dentro de la categoría File y en el Inspector de Objetos modifique sus propiedades como se indica en la siguiente tabla.

PROPIEDAD	VALOR
Caption	&Salvar
Hint	Salva un archivo
Name	SalvarArchivo

29. Seleccione la opción (No Category) dentro del listado de categorías del editor del Administrador de Acciones y agregue una nueva acción. En el Inspector de Objetos modifique las siguientes propiedades indicadas en la tabla.

PROPIEDAD	VALOR
Caption	&Acerca de
Category	Ayuda
Name	AyudaAcerca

30. Salve su proyecto.

Agregando un menú

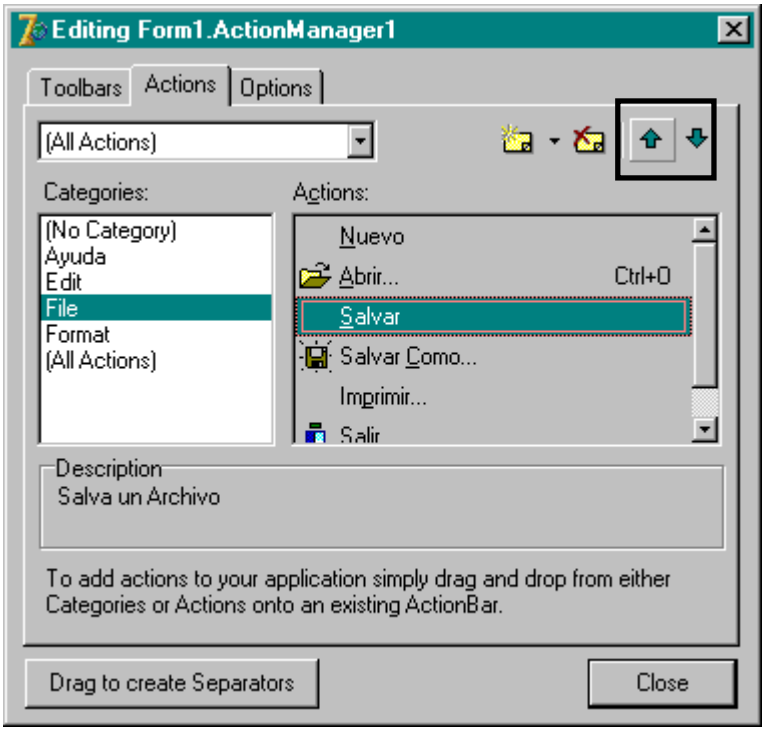
En esta sección agregaremos una barra de menú personalizada. La barra de menú de nuestro Editor de Texto, incluirá cuatro menús desplegables, Archivo, Edición, Formato y Ayuda con sus respectivos comandos.

El editor del Administrador de Acciones del componente TActionManager, facilita la creación de menús permitiendo arrastrar cada categoría y sus comandos a la barra de menú en un solo paso.

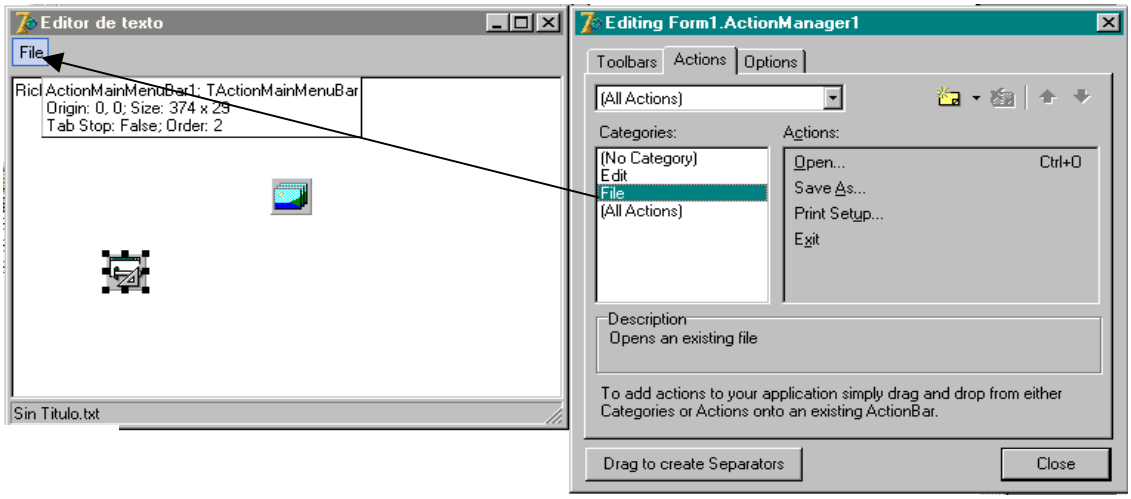
31. En la pestaña Additional de Paleta de Componentes, pulse doble click en el componente ActionMainMenuBar para agregarlo a la forma.

Una barra de menú en blanco aparecerá en la parte superior de la forma.

32. Abra el editor del Administrador de Acciones y seleccione la categoría File de la lista de Categorías, probablemente las acciones de esta categoría no se encuentren en el orden deseado, pero podemos modificar su orden, utilizando la combinación de teclas Ctrl + ↑ y Ctrl + ↓ o los botones Move Down y Move Up que se ubican en el extremo superior derecho del editor del Administrador de Acciones.



33. Modifique el orden de las acciones de la categoría File al orden: Nuevo, Abrir, Salvar, Salvar Como, Imprimir y Salir.
34. Arrastre la categoría File sobre el componente ActionMainMenuBar1 y observe como sus acciones aparecerán automáticamente formando el menú correspondiente, tal como se muestra en la siguiente figura.



35. Ahora, de la lista de categorías del Administrador de Acciones, arrastre una a una las categorías Edit, Format y Ayuda sobre el componente ActionMainMenuBar1 para completar el menú del Editor de Textos.

36. Dentro de la barra de menú seleccione el menú File y modifique su propiedad Caption al valor &Archivo, la propiedad Caption del menú Edit modifíquela al valor '&Edición' y finalmente modifique la propiedad Caption del menú Format por el valor &Formato.

37. Salve los cambios de su aplicación.

Agregando una barra de herramientas

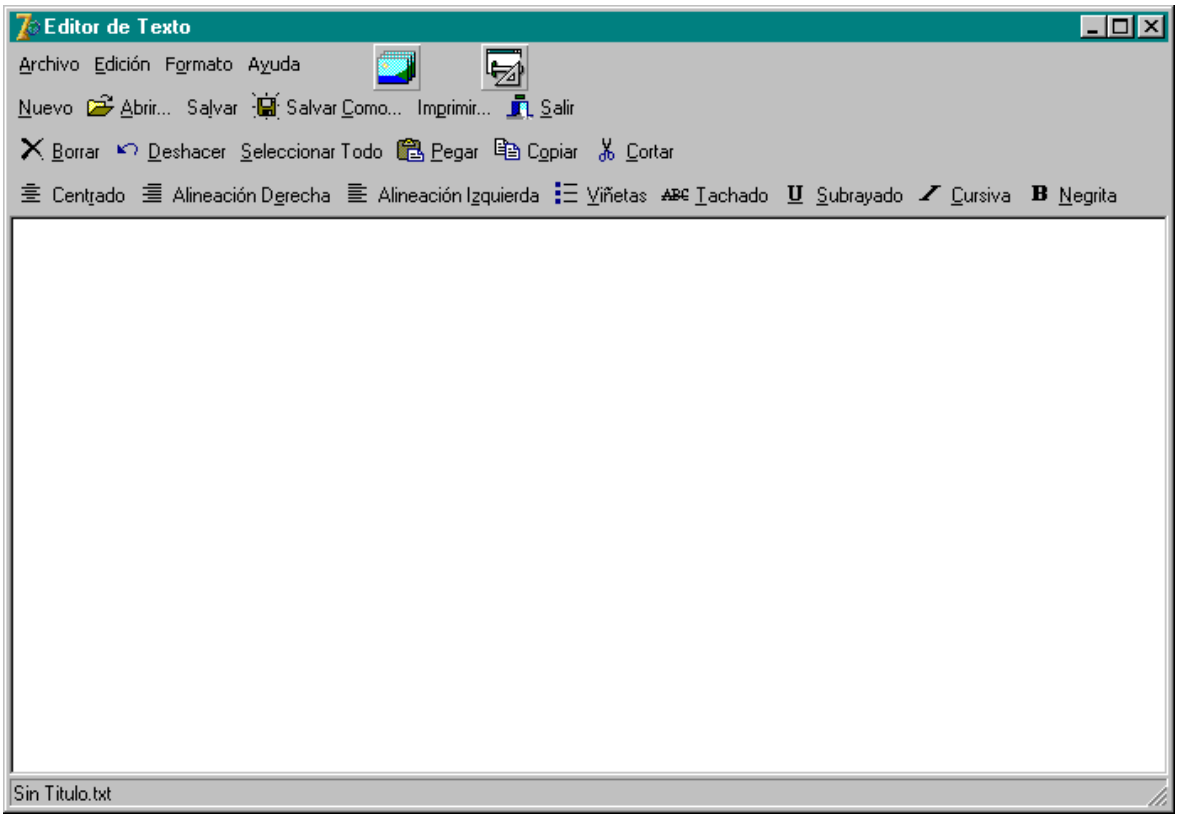
Ahora construiremos una barra de herramientas para proporcionar acceso fácil a los comandos de nuestra aplicación.

38. En la pestaña Additional de la Paleta de Componentes, pulse doble click en el componente ActionToolBar para agregarlo a la forma. Una barra de herramientas en blanco aparecerá debajo de la barra de menú.

39. Abra el editor del Administrador de Acciones y seleccione la categoría File de la lista de Categorías y arrástrela a la barra de herramientas. Los comandos de esta categoría aparecerán como botones en la barra de herramientas. De ser necesario modifique el orden de los botones para que su barra de herramientas tenga un orden.

40. De la pestaña Additional de la Paleta de Componentes, agregue dos componentes ActionToolBar más para generar las barras de herramientas para las categorías Edit y Format.

41. Desde el editor del Administrador de Acciones, arrastre las categorías Edit y Format a su componente ActionToolBar respectivo. Finalmente, su pantalla deberá tener el siguiente aspecto:



42. Cierre el editor del Administrador de Acciones y ejecute su proyecto (F9).

En este momento, el Editor de Texto ya posee algo de funcionalidad. Si selecciona texto en el área de edición, los botones Copiar, Cortar y Pegar deberán funcionar así como los botones correspondientes a las acciones de formato tales como Negrita, Cursiva, Subrayado, etc. Los menús y los botones de la barra de herramienta funcionan aunque algunos de los comandos están deshabitados, para activarlos debemos escribir manejadores de eventos.

Limpiando el área de edición

Cuando ejecutó el programa, el texto "RichEdit1" apareció en el área de texto, del componente RichEdit, se puede remover este texto por medio de la propiedad Lines del componente.

1. Cierre la ejecución de su proyecto y seleccione dentro de la forma principal el componente RichEditl.
2. En el Inspector de Objetos, localice la propiedad Lines, pulse doble click en el valor TStrings para desplegar el editor de String List.
3. En el editor de String List, seleccione y elimine el texto RichEditl y pulse click en OK.
4. Salve sus cambios.

Escribiendo manejadores de eventos

Hasta este punto hemos desarrollado la aplicación sin escribir una sola línea de código, utilizando solamente el Inspector de Objetos para configurar los valores de las propiedades de los componentes seleccionados, hemos aprovechado las ventajas del desarrollo RAD en Delphi.

En esta sección, escribiremos procedimientos llamados manejadores de eventos que responderán a las entradas de los usuarios mientras la aplicación se está ejecutando. Conectaremos los manejadores de eventos a los elementos de los menús y de la barra de herramientas, de tal manera que cuando se seleccione un elemento la aplicación ejecute la rutina del manejador de evento.

Para acciones no estándar, usted debe crear los manejadores de eventos. Para acciones estándar, como los comandos Archivo|Salir y Edición|Pegar. El código del manejador de eventos es generado automáticamente. Sin embargo, para algunas acciones estándar, tales como Archivo|Salvar podemos escribir un manejador de evento para personalizar el comando.

Debido a que todos los elementos del menú y de la barra de herramientas están consolidados en el Administrador de Acciones, podemos crear los manejadores de eventos desde ahí.

Creando un manejador de evento para el comando Nuevo

Para agregar un manejador de evento para el comando Nuevo, siga los siguientes pasos:

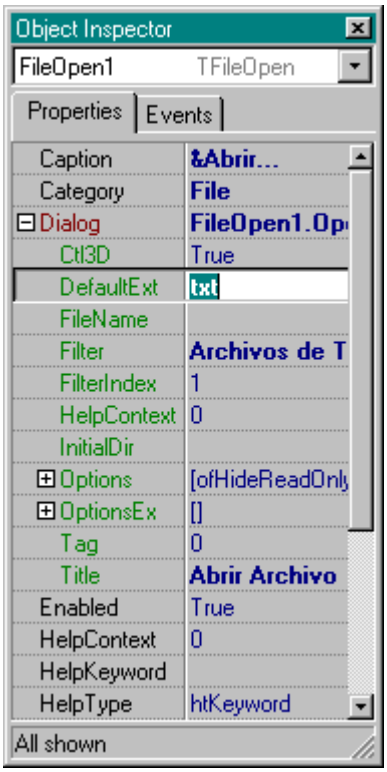
1. Seleccione el menú View | Units o con las teclas Ctrl+F12 y seleccione UMain para visualizar la unidad de código asociada a la forma de su editor.
2. Declare una variable llamada FileName de tipo String en la sección public de la clase Tforrn1.
3. Presione F12 para volver a la forma principal.
4. Pulse doble click en el componente ActionManager para abrir el editor del Administrador de Acciones.
5. Pulse doble click en la acción Nuevo, de la categoría File, Delphi generará el esqueleto del manejador de evento dentro de la unidad de código. Modifique el manejador de evento generado tal como se muestra en el siguiente listado:

```
procedure TForm1.ArchivoNuevoExecute(Sender: TObject);
begin
RichEdit1.Clear;
Filename:= 'Sin Titulo.txt';
StatusBar1.Panels[0].Text:=filename;
end;
```

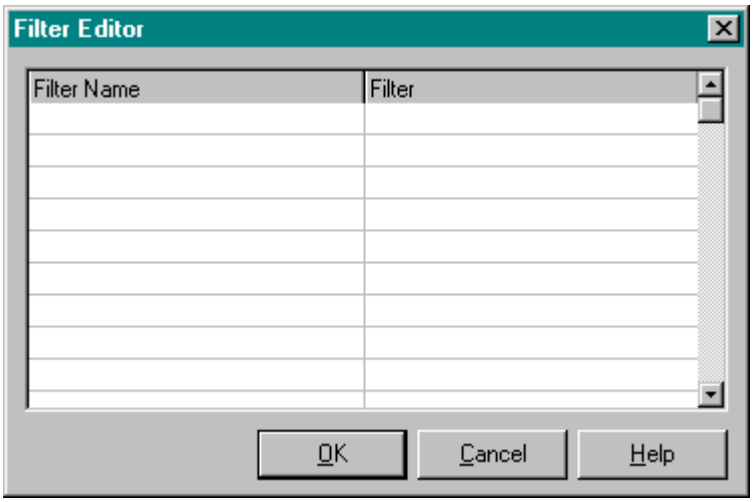
Creando un manejador de evento para el comando Abrir

Para abrir un archivo en el Editor de Texto hemos agregado un comando estándar: Abrir al Administrador de Acciones, este comando ya incluye un cuadro de diálogo abrir estándar de Windows, sin embargo debemos de personalizar el manejador de evento de este comando para que al seleccionar un archivo dentro del cuadro de diálogo su contenido se muestre dentro del componente RichEditl de nuestra forma.

1. Dentro del editor Administrador de Acciones seleccione la acción Abrir.
2. En el Inspector de Objetos pulse click en el signo + a la izquierda de la propiedad Dialog para expandir sus propiedades, Dialog es un componente referenciado que crea el cuadro de diálogo abrir. Delphi llama al cuadro de diálogo FileOpenl.Dialog por defecto. Cuando se ejecuta el método Execute del diálogo, se invoca el cuadro de diálogo estándar para abrir archivos.
3. Modifique la propiedad DefaultExt a txt

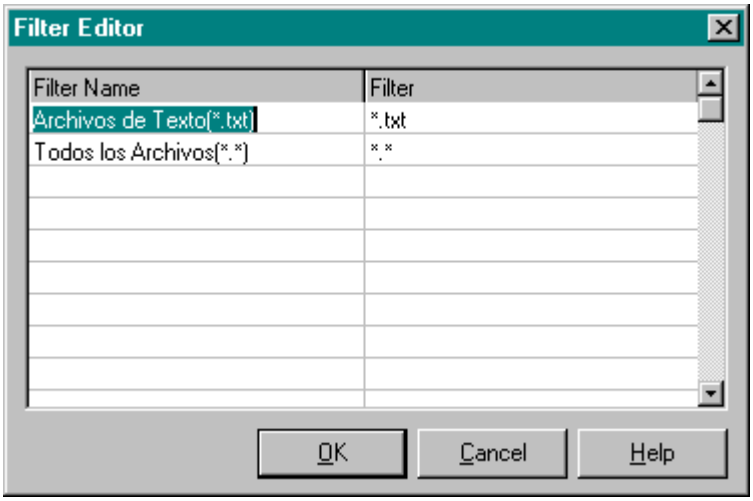


4. Pulse doble click en el área de texto del valor de la propiedad Filter para mostrar el editor de filtros.



5. Su En la primera fila, de la columna Filter Name, introduzca el valor Archivos de Texto (*.txt) y en la columna Filter introduzca el valor *.txt.
6. En la segunda fila, en la columna Filter Name, introduzca el valor Todos los Archivos (*.*) y en la columna Filter introduzca el valor *.*

El Editor de Filtros debe quedar tal como se muestra en la siguiente figura.



7. Pulse click en el botón OK.

8. Modifique la propiedad Title con el valor Abrir Archivo, esta frase aparecerán en la barra de titulo del cuadro de diálogo.
9. Pulse click en la pestaña Events del Inspector de Objetos.
10. Pulse doble click en el espacio a la derecha del evento OnAccept para que se genere el manejador de evento.
11. Modifique el manejador de evento como se muestra el siguiente listado.

```
procedure TForm1.FileOpen1Accept(Sender: TObject);
begin
   Richedit1.Lines.loadfromfile(fileopen1.dialog.filename);
   Filename:=fileopen1.dialog.filename;
   Statusbar1.panels[0].text:=filename;
end;
```

Creando un manejador de evento para el comando Salvar

1. Presione F12 para desplegar la forma y pulse doble click en el componente ActionManager.
2. Pulse doble click en la acción Salvar en el editor del Action Manager, el Editor de Código se abrirá con el cursor en el manejador de evento, complete el manejador de evento como se indica a continuación:

```
procedure TForm1.SalvarArchivoExecute(Sender: TObject);
begin
    if filename= 'Sin Titulo.txt' then
   filesaveas1.Execute
else
  richedit1.Lines.savetofile(filename);
end;
```

Este código le indica al editor de texto que muestre el cuadro de diálogo Salvar Como sí al archivo aun no se le ha dado un nombre. De otra forma, salva el archivo utilizando su nombre actual.

Creando un manejador de evento para el comando Salvar Como

Cuando el método Execute del cuadro de diálogo salvar se llama, este Involucra el cuadro de diálogo estándar de Windows Salvar Como para salvar archivos. A continuación crearemos un manejador de evento para el comando Salvar Como:

1. Presione la tecla de función F12 para desplegar la forma y pulse doble click en el ActionManager.
2. Seleccione la acción Salvar Como.
3. En el Inspector de Objetos, en la pestaña Propiedades. Pulse click en el signo + a la izquierda de la propiedad Dialog para expandir sus propiedades.
4. Modifique la propiedad DefaultExt a txt.

5. Pulse doble click en el área de texto de la propiedad Filter para desplegar el editor de filtros. En el editor de filtros especifique los siguientes valores:

En el primer renglón, en la columna Filter Name, introduzca el valor Archivos de Texto (*.txt), en la columna Filter introduzca el valor *.txt.

En el segundo renglón, en la columna Filter Name, introduzca el valor Todos los Archivos (".*), en la columna Filter introduzca el valor *.*.

6. Pulse click en el botón OK.
7. Modifique la propiedad Title al valor Salvar Como.
8. Pulse click en la pestaña Events del Inspector de Objetos, genere un manejador de eventos para el evento BeforeExecute y agregue la siguiente línea al manejador de eventos:

```
procedure TForm1.FileSaveAs1BeforeExecute(Sender: TObject);
begin
  filesaveas1.Dialog.InitialDir:=extractfilepath(filename);
end;
```

9. Genere un manejador de evento para el evento OnAccept y modifique el manejador de evento tal como se muestra en el siguiente listado.

```
procedure TForm1.FileSaveAs1Accept(Sender: TObject);
begin
  filename:=filesaveas1.Dialog.FileName;
  richedit1.lines.savetofile(filename);
  statusbar1.Panels[0].Text:=filename;
end;
```

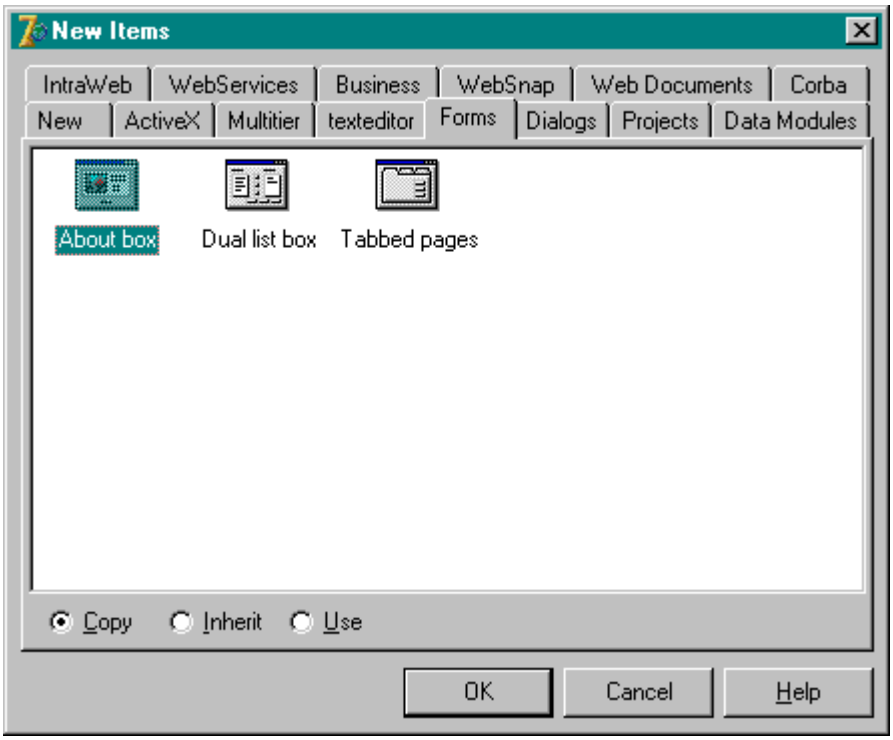
10. Salve su proyecto.

Creando un Cuadro de Diálogo Acerca de

Muchas aplicaciones incluyen un cuadro de diálogo de Acerca de, el cual despliega información del producto, como el nombre, versión, logotipos, y puede incluir información legal, incluyendo información de derechos de autor.

Ya hemos creando un comando Acerca de, en el Action Manager, a continuación generaremos un manejador de eventos para agregar un cuadro de diálogo Acerca de en nuestro Editor de Textos:

1. Seleccione File | New | Other del menú de Delphi, para desplegar el cuadro de diálogo New Items.
2. Seleccione la pestaña Forms y pulse doble click en el icono About box. como se muestra en la siguiente figura.



Dentro de su aplicación aparecerá una forma prediseñada Acerca de.

3. Seleccione la forma, y modifique su propiedad Caption al valor Acerca del Editor de Texto.
4. Modifique las propiedades de los siguientes componentes a través del Inspector de Objetos como se indica en la tabla:

COMPONENTE	PROPIEDAD	VALOR
ProductName	Caption	Editor de Texto
Versión	Caption	Versión 1.0
Copyright	Caption	Copyright 2003

5. Salve el archivo del cuadro de diálogo Acerca de cómo About.pas
6. Selecciónela unidad UMain.
7. Del menú de Delphi seleccione File | Use Unit y en la lista de unidades del cuadro de diálogo Use Unit seleccione la unidad About, para agregar la unidad About a la cláusula uses de la unidad UMain.
8. Presione la tecla F12 para mostrar la forma, pulse doble click en el Action Manager y pulse doble click en la acción Ayuda | Acerca de para crear un manejador de evento, en él, agregue la siguiente línea al manejador del evento;

```
procedure TForm1.AyudaAcercaExecute(Sender: TObject);
var
```

```
 fAbout:tAboutbox;
begin
   fAbout := taboutbox.Create(nil);
try
   Fabout.ShowModal;
finally
    Fabout.Free;
end;
end;
```

Este código, abre el cuadro de diálogo Acerca de cuando el usuario selecciona el menú Ayuda | Acerca de, dentro del editor de textos. El método ShowModal abre la forma en un modo modal, en donde el usuario no puede hacer nada en la aplicación hasta que cierra la forma.

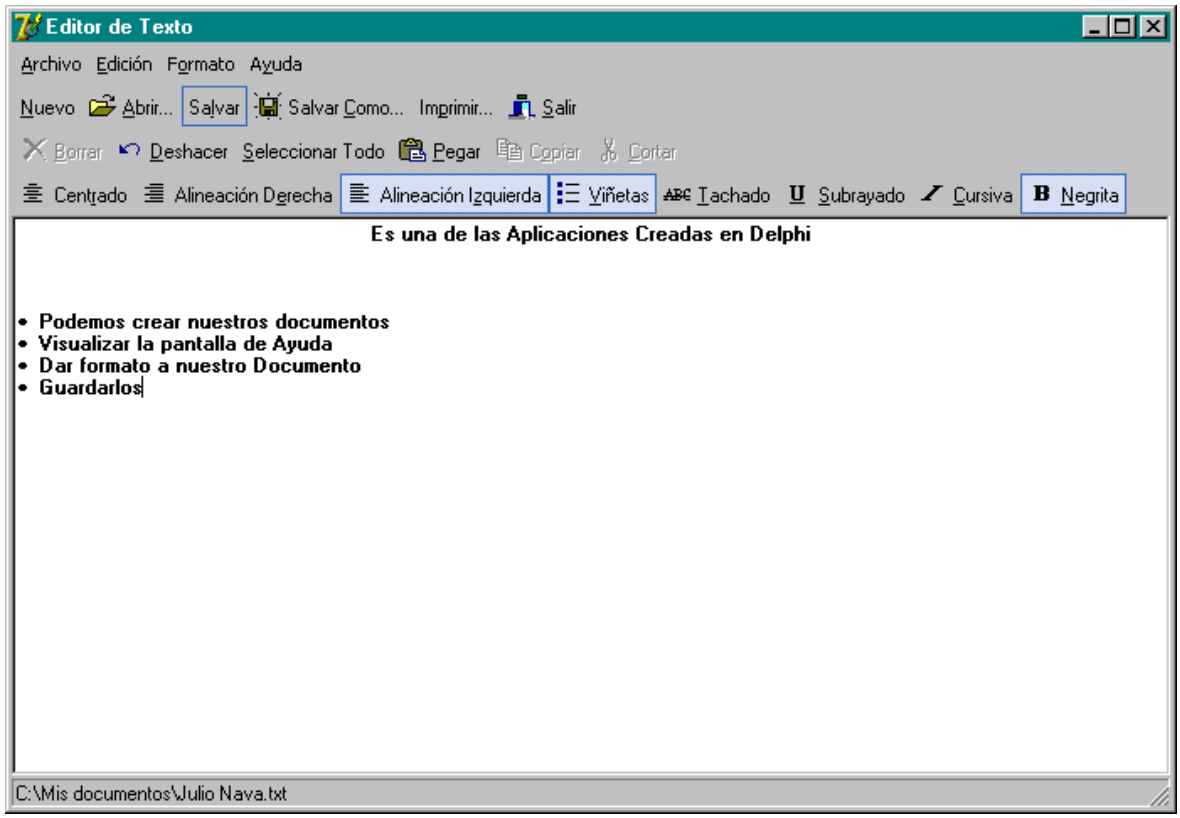
Completando la aplicación

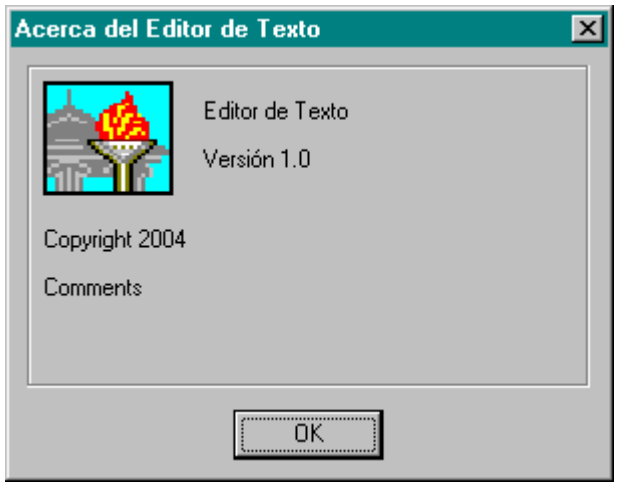
La aplicación está casi completa, para finalizar la aplicación:

1. Presione la tecla de función F12 para mostrar la forma principal.
2. Seleccione la forma a través del Inspector de Objetos, seleccione la pestaña Events y pulse doble click en el área de texto del evento OnCreate.
3. Agregue la siguiente línea de código al manejador del evento.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  archivonuevo.Execute;
end;
end.
```

4. Salve su proyecto.
5. Ejecute su aplicación (F9) y pruébela.


Editor de Texto
Archivo Edición Formato Ayuda
Nuevo Abrir... Salvar Salvar Como... Imprimir... Salir
Borrar Deshacer Seleccionar Todo Pegar Copiar Cortar
Centrado Alineación Derecha Alineación Izquierda Viñetas Tachado Subrayado Cursiva Negrita
Es una de las Aplicaciones Creadas en Delphi
Podemos crear nuestros documentos
Visualizar la pantalla de Ayuda
Dar formato a nuestro Documento
Guardarlos
C:\Mis documentos\Julio Nava.txt


Acerca del Editor de Texto
Editor de Texto
Versión 1.0
Copyright 2004
Comments
OK

7.3 Preguntas y Ejercicio de Repaso

1. Describa el funcionamiento del proceso Drag and Drop.

2. ¿La propiedad DragMode para que me sirve y dime sus dos valores que puede tomar?

3. Describa la función del evento OnkeyPress.

4. Describa la función del componente TXPManifest.

5. Describa la función del componente TActionManager.

6. Explica para que me sirve el siguiente codigo

```
procedure TForm1.edalfabeticoKeyPress(Sender: TObject; var Key: Char);
begin
{#8 BkSp
#l3 Enter
#18 Tab}
if not (key in ['a'..'z', 'A'..'Z', #8, #13, #18]) then
   Key := Char(0);
end;
```

7. Describe los componentes TimageList y ActionManager

VII. Tecnologías de Acceso a Datos

8.1 Introducción.

"Una de las principales características de Delphi es su soporte para el desarrollo de aplicaciones de bases de datos. En las primeras versiones de Delphi, la única tecnología de acceso a datos disponible era el Borland Database Engine (BDE). A partir de Delphi 3, el código de la VCL relacionado con el acceso a bases de datos fue reestructurado para permitir más opciones para el acceso a datos. Delphi 5 agregó dos conjuntos de componentes para soportar el acceso a datos a través de ActiveX Data Objects (ADO) de Microsoft e Interbase Express (IBX).

En Delphi 6 se agregó soporte a dbExpress, el cual es una tecnología de acceso a datos de desarrollo interplataforma en Windows y Linux, además de ser independiente de la base de datos.

En Delphi 7 Borland ha marcado como obsoletos los SQL Links (los cuales describiremos más adelante), además de agregar soporte para Informix SE, Oracle 9i, DB2 7.2, Interbase 6.5 y MySQL 3.23.49.

A continuación describiremos las distintas alternativas de las tecnologías de acceso a datos que se pueden utilizar en Delphi 7 para desarrollar aplicaciones de bases de datos."[1]

8.2 Borland Database Engine (BDE)

La tecnología de BDE se originó antes de la existencia de Delphi, utilizando Paradox, poco después Borland le agregó soporte para otras bases de datos locales como FoxPro y Access; y servidores de SQL como Oracle, Sybase, Informix, DB2, el acceso a servidores de SQL se realiza a través de manejadores llamados SQL Links.

La ventaja de utilizar un motor de bases de datos común es que las aplicaciones pueden ser más portables entre distintos servidores del mismo tipo. La tecnología de BDE tiene la ventaja de que está fuertemente integrada en Delphi además de que es la única solución viable para el acceso a bases de datos de archivos locales como Paradox y DBase.

Sin embargo, Borland ha dejado de desarrollar la tecnología de BDE y no existirán más actualizaciones, por lo que se recomienda que las aplicaciones de bases de datos que accedan a servidores de SQL a través de BDE, se migren a otra tecnología de acceso a datos como dbExpress.

[1] Ayuda del Software Delphi 7

8.2.1 Componentes BDE

Los componentes de acceso a datos que utilizan el Borland Database Engine se encuentran en la pestaña BDE de la paleta de componentes y se listan a continuación.

TTable

Encapsula una tabla de una base de datos

La siguiente tabla muestra las propiedades de la clase TTable.

PROPIEDAD	DESCRIPCIÓN
CanModify	Indica si una aplicación puede insertar, editar y eliminar información en una tabla.
DataSource	Proporciona acceso de solo lectura al dataset maestro cuando la tabla es el detalle de una relación maestro/detalle.
Defaultindex	Especifica si los datos en la tabla deberían estar ordenados por un índice por defecto cuando se abre.
Exclusive	Permite a una aplicación obtener acceso total a una tabla de Paradox o DBase.
Exists	Indica si la tabla de la base de datos existe.
IndexDefs	Contiene información acerca de los índices de una tabla.
IndexFieldCount	Indica el número de campos que forman la llave actual.
IndexFieldNames	Lista las columnas que se utilizan en un índice para una tabla.
IndexFields	Lista los campos del índice actual.
IndexFiles	Especifica uno o más archivos de índice de DBASE
IndexName	Identifica un índice secundario de una tabla.
KeyExclusive	Especifica como deben interpretarse el límite superior e inferior de un rango.
KeyFieldCount	Especifica el número de campos a utilizar cuando se lleva a cabo una búsqueda de llave parcial en una llave de múltiples campos.
MasterFields	Especifica uno o más campos en una tabla maestro a ligar con los campos correspondientes en esta tabla para establecer una relación maestro/detalle entre las tablas.
MasterSource	Especifica el nombre del data source para un data set a utilizar como tabla maestra al establecer una relación maestro/detalle para esta tabla y otra.
ReadOnly	Especifica si una tabla es de solo lectura.

StoreDefs	Indica si los campos de la tabla y las definiciones de índices persisten en el datamodule o forma.
TableLevel	Especifica el nivel de dependencia de la tabla y el driver de BDE.
TableName	Indica el nombre de la tabla de la base de datos que encapsula este componentes.
TableType	Indica la estructura de la tabla de la base de datos que este componente representa.

TQuery.

Representa un conjunto de datos (dataset) que está basado en una sentencia SQL.

Utilice TQuery para acceder a una o más tablas en una base de datos utilizando sentencias SQL, los componentes Query se pueden utilizar con servidores remotos de bases de datos, tales como Sybase, SQL Server, Oracle, Informix, DB2, e Interbase, con tablas locales: Paradox, Interbase, dBASE, Access y FoxPro y con bases de datos que soporten ODBC.

Los componentes Query son útiles debido a que pueden:

- Acceder a más de una tabla a la vez (un "join" de SQL)
- Automáticamente acceder a un subconjunto de registros y columnas de las tablas de la base de datos, en lugar de siempre devolver todos los registros y columnas.

A continuación se listan las propiedades del componente TQuery.

PROPIEDAD	DESCRIPCIÓN
Constrained	Indica si las actualizaciones e inserciones que no se adecuen al conjunto generado por una sentencia SELECT se permiten para las tablas de DBASE o Parados.
DataSource	Especifica el componente data source asociado a un componente Query Maestro desde el cual se extraen los valores actuales de los campos para utilizar en los parámetros con los mismos nombres en la sentencia SQL de un Query Detalle.

Local	Indica si el Query hace referencia a una o más tablas de Paradox o DBASE en lugar de tablas de SQL en un servidor remoto.
ParamCheck	Especifica si la lista de parámetros para un Query es regenerada si la propiedad SQL se modifica en tiempo de ejecución.
ParamCount	Indica el número actual de parámetros para el Query.
Params	Contiene la lista de parámetros para la sentencia SQL de un Query.
Prepared	Determina si el Query está preparado o no para su ejecución.
RequestLive	Solicita un conjunto de datos actualizable desde el Query.
RowsAffected	Devuelve el número de registros afectados por la última ejecución del Query.
SQL	Contiene el texto de la sentencia SQL a ejecutar por el Query.
SQLBinary	Apunta a los datos binarios que representan la sentencia SQL del Query o conjunto resultante.
Text	Apunta al texto actual de SQL que el Query pasa al BDE.
UniDirectional	Determina si los cursores bidireccionales de BDE están habilitados para el conjunto resultante del Query.

TStoredProc

Enacapsula un procedimiento almacenado en una aplicación basada en BDE.

Utilice un objeto TStoredProc en aplicaciones basadas en BDE para utilizar un procedimiento almacenado en un servidor de bases de datos. Un procedimiento almacenado es un conjunto de sentencias agrupadas, almacenado como parte de los metadatos del servidor de bases de datos (como las tablas, índices y dominios), que ejecuta una tarea relacionada con la base de datos en el servidor y que pasa resultados al cliente.

La tabla siguiente lista las propiedades del componente TStoredProc.

PROPIEDAD	DESCRIPCIÓN
Overload	Especifica que procedimiento almacenado sobrecargado, de Oracle ejecutar.
ParamBindMode	Determina el orden en el cual los parámetros de un componente son asignados a la lista de parámetros del procedimiento almacenado en el servidor.

ParamCount	Indica el número de parámetros del procedimiento almacenado.
Params	Almacena los parámetros de entrada y salida del procedimiento almacenado.
Preparad	Determina si el procedimiento almacenado está listo para su ejecución.
StorecProcName	Identifica el nombre del procedimiento almacenado en el servidor.

TDatabase.

Proporciona control sobre la conexión a una sola base de datos en aplicaciones de bases de datos basadas en BDE.

Utilice TDatabase cuando una aplicación de bases de datos basada en BDE requiera realizar las siguientes acciones sobre una conexión a base de datos:

- Personalizar autentificación al servidor de bases de datos
- Conexiones persistentes a bases de datos
- Control de transacciones
- Alias específicos de BDE

TDatabase es importante por el control que proporciona sobre procesamiento de transacciones con el BDE cuando está conectado a un servidor SQL remoto.

A continuación se listan las propiedades del componente TDatabase.

PROPIEDAD	DESCRIPCIÓN
AliasName	Especifica un alias de BDE utilizado por esta conexión.
Connected	Indica si la conexión de base de datos está activa.
DatabaseName	Especifica el nombre de la base de datos asociada con este componente.
DataSets	Proporciona un array indexado de todos los datasets activos para un componente database.
Directory	Especifica el directorio de trabajo para una base de datos DBASE o Paradox.
DriverName	Especifica el nombre del driver de BDE para la base de datos.
Exclusive	Permite que una aplicación obtenga acceso exclusivo sobre una base de datos.

Handle	Especifica el handle de base de datos de BDE.
HandIeShared	Especifica si se comparte un handle de base de datos.
InTransaction	Indica si una transacción de base de datos está en progreso o no.
IsSQLBased	Indica si un componente database está utilizando ya sea un driver de SQL Links de BDE o el socket ODBC de BDE.
KeepConnection	Especifica si la aplicación permanece conectada a la base de datos aunque no existan datasets abiertos.
Locale	Identifica el lenguaje del driver de BDE para el componente database.
Params	Contiene los parámetros de conexión a base de datos para el alias de BDE asociado con el componente database.
Session	Apunta al componente session al que está asociado este componente.
SessionAlias	Especifica si el componente database está utilizando un alias de sesión.
SessionName	Especifica el nombre de la sesión utilizada por este componente.
ReadOnly	Especifica si la conexión de base de datos proporciona acceso de sólo lectura.
Temporary	Indica si el componente database es temporal o uno persistente.
TraceFlags	Especifica las operaciones de base de datos a seguir con SQL monitor en tiempo de ejecución.
TransIsolation	Especifica el nivel de aislamiento de transacciones manejado por el BDE.

Tsession

Proporciona la administración global de un grupo de conexiones a bases de datos en una aplicación.

Utilice TSession para administrar un grupo de conexiones a bases de datos dentro de una aplicación. Existen tres usos para TSession: Estándar, múltiples archivos de red para Paradox y aplicaciones de múltiples capas.

La librería automáticamente crea un componente global TSession por defecto llamado Session para todas las aplicaciones de base de datos. El componente por defecto session maneja las conexiones estándar de bases de datos. Una aplicación puede controlar la sesión por defecto accesando a sus propiedades, eventos y métodos en tiempo de ejecución.

Las aplicaciones de bases de datos que deban de acceder simultáneamente a tablas de paradox ubicadas en diferentes localidades de la red pueden establecer múltiples sesiones, una para cada localidad de la red.

Finalmente, las aplicaciones de bases de datos que deben establecer múltiples conexiones concurrentes a la misma base de datos al mismo tiempo, tales como realizar dos queries contra los mismos datos al mismo tiempo, son aplicaciones de múltiples capas.
La tabla siguiente muestra las propiedades del componente TSession

PROPIEDAD	DESCRIPCIÓN
Active	Especifica si la conexión está o no activa.
AutoSessionName	Indica si un nombre único de sesión se genera automáticamente para cada sesión.
ConfigMode	Especifica como el BDE debería de manejar los alias para la sesión.
DatabaseCount	Indica el número de componentes database activos actualmente asociados con la sesión.
Databases	Proporciona un arreglo de todos los componentes database activos nombrados para una sesión.
Handle	Especifica el handle de BDE para la sesión.
KeepConnections	Especifica si un componente database temporal, creado en el contexto de una sesión, mantiene una conexión al servidor de bases datos, aunque no existan datasets activos asociados con el componente database.
Locale	Identifica el lenguaje del driver de BDE para el componente session.
NetFileDir	Específica el directorio que contiene el archivo de control de red PDOXUSRS.NET
PrivateDir	Especifica el directorio en el que se almacenan los archivos de procesamiento temporal de tablas generados por el BDE para componentes database asociados con una sesión.
SessionName	Especifica un nombre único de sesión que puede ser utilizado por componentes database y dataset para ligarse a esta sesión.
SQLHourGlass	Indica si el cursor del mouse cambia o no a un reloj de arena durante operaciones del BDE.
TraceFlags	Especifica las operaciones de bases de datos a rastrear por el SQL Monitor en tiempo de ejecución.

TBatchMove

Realiza operaciones de bases de datos en grupos de registros o tablas enteras.

Utilice un componente TBatchMove para:

- Agregar los registros de un dataset a una tabla de base de datos.
- Eliminar los registros de un dataset de una tabla de bases de datos.
- Copiar un dataset para crear una nueva tabla de base de datos o sobrescribir una tabla existente.

Modifique la propiedad Mode para especificar la operación deseada. Las propiedades Source y Destination indican los datasets cuyos registros se agregarán eliminarán o copiarán. Otras propiedades especificarán como realizar la operación y como manejar problemas que TBatchMove puede encontrar cuando realiza la operación. Una vez que se han establecido las propiedades para indicar la operación a realizar, llame al método Execute para realizar la operación.

A continuación se listan las propiedades definidas por el componente TBatchMove.

PROPIEDAD	DESCRIPCIÓN
AbortOnKeyViol	Especifica si la operación se termina cuando se detecta una violación de integridad de llave.
AbortOnProblem	Especifica si la operación se termina inmediatamente cuando es necesario truncar la información para colocarla en el destino especificado.
ChangedCount	Indica el número de registros de la tabla de destino que son alterados como resultado de la operación.
ChangedTableName	Especifica el nombre de la tabla local de Paradox que se crea para almacenar las copias de todos los registros de la tabla de destino modificada por la operación.
CommitCount	Especifica cuantos registros son modificados antes de que ocurra un commit.
Destination	Especifica el objeto Ttable para la tabla de base de datos que es el destino de la operación.
KeyViolCount	Reporta el número de registros que no pudieron ser reemplazados, agregados o eliminados del destino debido a violaciones de integridad de llaves.

KeyViolTableName	Especifica el nombre de la tabla de Paradox que será creada para almacenar todos los registros de la fuente que no podrán participar en la operación debido a violaciones de integridad de llaves.
Mappings	Especifica el mapeo de columnas para la operación.
Mode	Especifica la operación a ejecutar cuando se invoque la operación Execute.
MovedCount	Reporta el número de registros de la fuente que fueron aplicados al destino.
ProblemCount	Indica el número de registros que no pudieron ser agregados a! destino sin perder información debido a tipos incompatibles de datos.
ProblemTableName	Especifica el nombre de la tabla de Paradox que será creada para almacenar todos los registros de la fuente que contengan campos que serán recortados para coincidir con el tipo de campo de la tabla de destino.
RecordCount	Específica el número máximo de registros que son aplicados al destino cuando se llame Execute.
Source	Especifica el dataset que es la fuente la operación.
Transliterate	Especifica si los datos en los registros fuente deberían ser convertidos de la localidad de la fuente a la localidad del destino cuando se invoque el método Execute.

TUpdateSQL

Aplica las actualizaciones en cache que los queries o stored procedures no pueden actualizar directamente.

Utilice un objeto TUpdateSQL para proporcionar sentencias SQL que permite actualizar datasets de solo lectura representados por componentes TQuery o TstoredProc cuando las actualizaciones de caché están habilitadas. Un dataset es de solo lectura ya sea por diseño o circunstancia. Si un dataset es de solo lectura por diseño, la aplicación no debe proporcionar una interfaz al usuario para actualizar la información, pero puede proporcionar un esquema programático tras bambalinas. Si un dataset es de solo lectura por alguna circunstancia, el BDE indica que es de solo lectura, esto usualmente ocurre con queries realizados contra múltiples tablas, tales queries son por la definición SQL-92 de solo lectura.

TUpdateSQL proporciona un mecanismo para evitar lo que algunos desarrolladores consideran una limitación del SQL-92. Permite proporcionar sentencias INSERT, UPDATE y DELETE para realizar queries separados en datasets que son de solo lectura, de tal forma que estos queries separados son transparentes para el usuario final.

La tabla siguiente muestra las propiedades definidas por el componente TUpdateSQL.

PROPIEDAD	DESCRIPCIÓN
DatabaseName	Identifica la base de datos a la que se están aplicando las actualizaciones.
DataSet	Identifica el dataset que almacena la información actualizada.
DeleteSQL	Especifica la sentencia de SQL DELETE a utilizar cuando se aplique la eliminación de caché a un registro.
InsertSQL	Especifica la sentencia de SQL INSERT a utilizar cuando se aplique la inserción de caché a un registro.
ModifySQL	Especifica la sentencia de SQL UPDATE a utilizar cuando se aplique la actualización de caché a un registro.
Query	Devuelve el objeto Query utilizado para realizar un tipo especifico de actualización.
SessionName	Identifica la sesión bajo la cual se aplican las actualizaciones.
SQL	Devuelve una sentencia SQL especificada cuando se apliquen las actualizaciones de caché.

TNestedTable.

Encapsula un dataset que está anidado como campo dentro de otra tabla.

Utilice TNestedTable para acceder datos contenidos en un dataset anidado. Una tabla anidada hereda funcionalidad de BDE de TBDEDataSet y por lo tanto utiliza el Borland Database Engine (BDE) para obtener acceso a la tabla anidada. Una tabla anidada proporciona mucha de la funcionalidad de un componente table, excepto que la información a la que accede está almacenada en una tabla anidada.

La siguiente tabla lista las propiedades del componente TnestedTable

PROPIEDAD	DESCRIPCIÓN
BlockReadSize	Determina cuantos registros de buffer se leen en cada bloque.
CacheBlobs	Determina si los campos BLOB se almacenan en memoria cache.
CachedUpdates	Determina si las actualizaciones en cache se habilitan para un dataset.
CanModify	Especifica si la base de datos otorga permiso de escritura a los datos.
ExpIndex	Indica si un dataset está utilizando una expresión de índice de DBASE.
Filter	Especifica el texto del filtro actual de un dataset.
Filtered	Especifica sí el filtrado está activo para un dataset.
FilterOptions	Especifica si el filtrado es sensible a mayúsculas y minúsculas, y si se permiten las comparaciones parciales cuando se están filtrando registros.
Handle	Especifica el handle de cursor de BDE para el dataset.
KeySize	Especifica el tamaño de la llave para e! índice actual del dataset.
Locale	Identifica el driver de lenguaje de BDE para el dataset.

PROPIEDAD	DESCRIPCIÓN
RecNo	Indica el registro actual en el dataset.
RecordCount	Indica el número total de registros asociados con el dataset.
RecordSize	Indica el tamaño de un registro en el dataset.
UpdateObject	Especifica el componente update utilizado para actualizar un resultado de solo lectura cuando las actualizaciones de cache están habilitadas.
UpdateRecordTypes	Especifica el tipo de registros visibles en un dataset cuando las actualizaciones de cache están habilitadas.
UpdatesPending	Indica sí el buffer de actualizaciones de cache contiene registros que aún no han sido aplicados.

TNestedTable solamente define un método, el método Create el cual crea una nueva instancia del componente TNestedTable.

"Desarrollo de una aplicación utilizando el BDE."[2]

1. Cree una nueva aplicación File | New | Application, salve su proyecto con los siguientes nombres de archivo:

Archivo de la forma: UMain.pas

Archivo del proyecto: BDEDemo.dpr

[2] http://www.delphi3000.com/

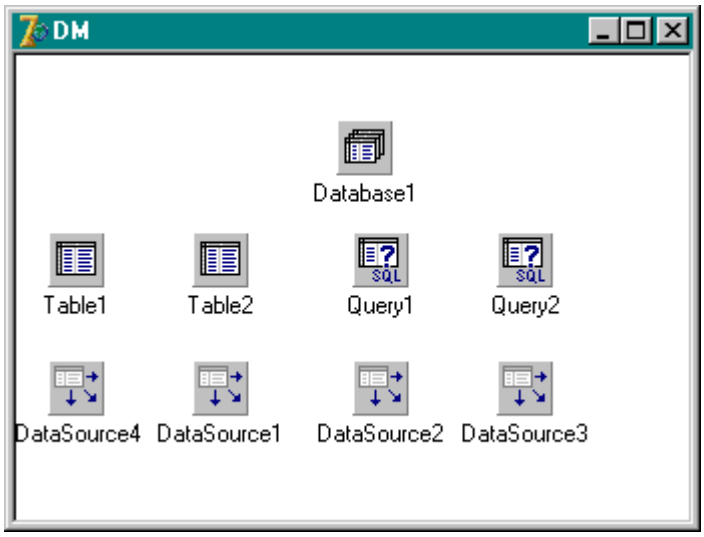
2. A continuación agregaremos un Data Module a nuestra aplicación. Un Data Module es un tipo especial de forma donde se pueden centralizar los componentes de acceso a datos; para crear el Data Module, seleccione del menú de Delphi la opción File | New | Data Module,
3. Salve el archivo del Data Module con el nombre UDataModule.pas.
4. Modifique la propiedad Name del Data Module al valor DM.
5. Coloque los siguientes componentes dentro del Data Module:

Un Componente TDatabase (pestaña BDE)

Dos componentes TTable (pestaña BDE)

Dos componentes TQuery (pestaña BDE)

Cuatro componentes TDataSource (pestaña DataAccess)



6. Modifique las propiedades de dichos componentes como se muestra en la siguiente tabla:

COMPONENTE	PROPIEDAD	VALOR
Database1	AliasName	DBDEMOS
	DatabaseName	Conexión Dbdemos
	KeepConnection	False
	LoginPrompt	False
	Name	DBDbdemos
	Connected	True

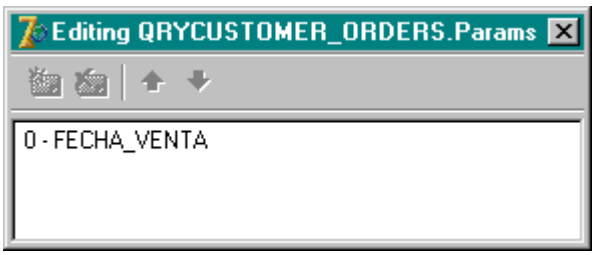
Tablel	Name	TbCustomer
	DatabaseName	Conexión Dbdemos
	TableName	CUSTOMER.DB
	Active	True
DataSourcel	Name	DSCustomer
	DataSet	TbCustomer
Table2	Name	TbOrders
	DatabaseName	Conexión Dbdemos
	TableName	ORDERS.DB
	MasterSource	DsCustomer
	MasterFields	Custno -> CustNo
	Active	True
DataSource2	Name	DSOrders
	DataSet	TbOrders
Queryl	Name	QryCustomer_City
	DatabaseName	Conexión Dbdemos
	SQL	Select
		DISTINCT(Country)
		from Customer
	Active	True
DataSource3	Name	DsCustomer_Orders
	DataSet	QryCustomer_Orders
Query2	Name	QryCustomer_Orders
	DatabaseName	Conexión Dbdemos
	SQL	Select C.CustNo,
		C.Company, C.Phone,
		O.OrderNo, O.ShipDate,
		O.AmountPaid
		From CustomerC,
		Orders O
		Where C.Custno = O.Custno
		and O.ShipDate >=:Fecha Venta
DataSource4	Name	DsCustomer_Orders2
	DataSet	QryCustomer_Orders

El componente Database proporciona un control sobre las conexiones a una base de datos que se emplean en la aplicación.

El objetivo de los componentes TTable (TbCustomer y TbOrders), que se colocaron dentro del Data Module, es crear una relación Maestro/Detalle entre las tablas de Clientes y Ordenes de la base de datos DBDEMOS.

El componente QryCustomer_Orders declara, a través de su propiedad SQL, una sentencia SQL parametrizada; los parámetros se declaran con el prefijo ':'. Y para que el query sea funcional, debemos de especificar el tipo de dato del parámetro, para lo cual:

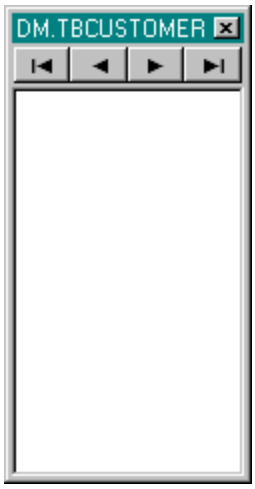
7. Pulse click en la propiedad Params del componente QryCustomer_Orders, aparecerá el editor de parámetros como se muestra en la siguiente figura.



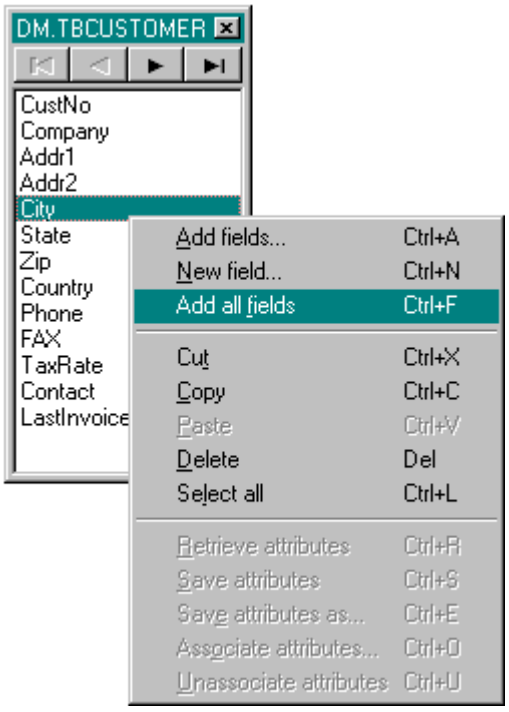
8. Modifique la propiedad DataType del parámetro al valor ftDate. Cierre el cuadro de diálogo del editor de parámetros.

9. Modifique la propiedad Active del componente QryCustomer_Orders al valor True.

10. Pulse doble click sobre el componente TbCustomer para visualizar el editor de campos, el cual se muestra en la siguiente figura:



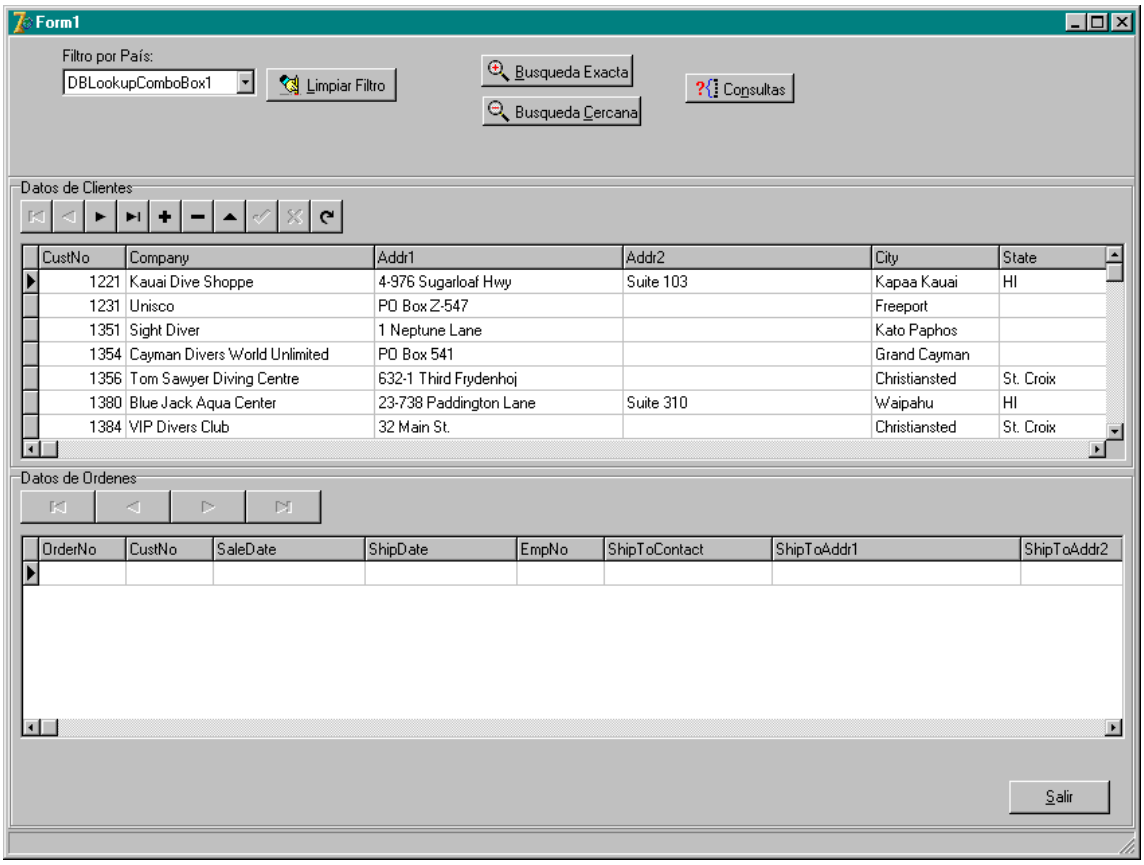
11. Pulse click con el botón derecho del mouse sobre el editor de campos y seleccione la opción Add all Fields del menú contextual que se presenta para agregar todos los campos de la tabla Customer dentro de editor de campos.



12. Agregue dentro del Editor de campos de cada uno de los componentes DataSet (Table y Query) sus campos correspondientes, de la misma manera como se realizó con el componente TbCustomer.

13. Grabe los cambios de su proyecto.

A continuación configuraremos la forma principal del proyecto, como se muestra en la siguiente figura.

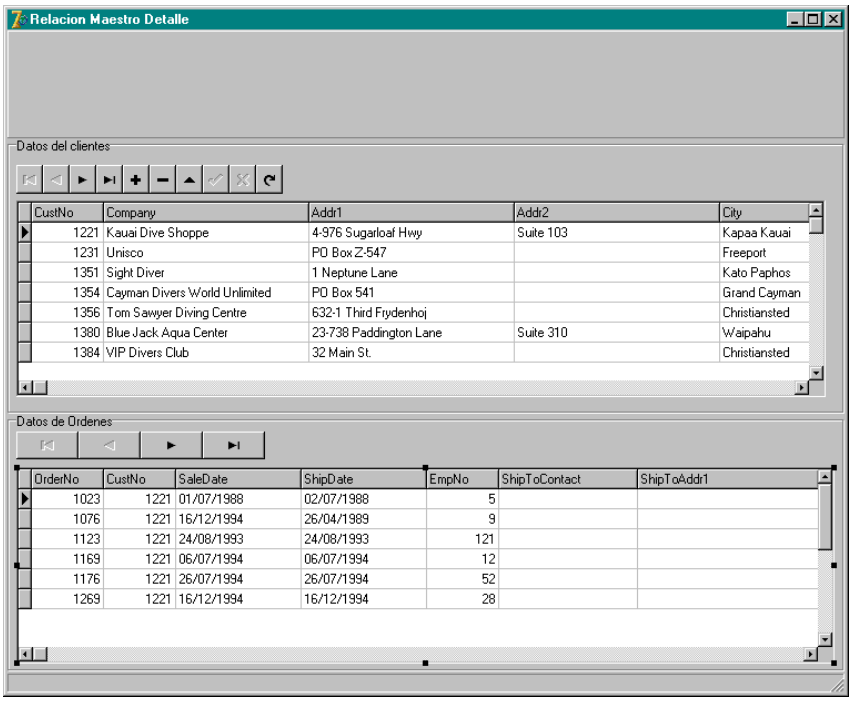


14. Modifique la propiedad Name de la forma al valor FrmMain y su propiedad Caption al valor Relación Maestro Detalle.
15. Agregue a la forma un componente TStatusBar (pestaña Win32).
16. Del menú de Delphi seleccione la opción File | Use Unit y seleccione la unidad UDataModule dentro del cuadro de diálogo Use Unit. Pulse click en el botón OK.
17. Agregue a la forma un componente TPanel (pestaña Standard) modifique su propiedad Align al valor alTop y elimine el contenido de su propiedad Caption.
18. Agregue en la forma un componente TGroupBox (pestaña Standard) y modifique su propiedad Align al valor alTop y su propiedad Caption a Datos de Clientes.

19. Agregue dentro del componente GroupBoxl un componente TDBNavigator (pestaña Data Controls), colóquelo en la parte superior del componente GroupBoxl y modifique su propiedad DataSource al valor DM.DsCustomer.
20. Dentro del componente GroupBoxl agregue un componente TDBGrid (pestaña Data Controls) y modifique su propiedad DataSource al valor DM.DsCustomer.
21. Agregue otro componente TGroupBox (pestaña Standard) en la forma y modifique su propiedad Align al valor alClient y su propiedad Caption a Datos de Ordenes.
22. Agregue dentro del componente GroupBox2 un componente TDBNavigator (pestaña Data Controls), colóquelo en la parte superior del componente GroupBox2 y modifique su propiedad DataSource al valor DM.DsOrders.
23. Modifique la propiedad VisibleButtons del componente DBNavigator2 como se muestra a continuación.

SUBPROPIEDAD	VALOR
NbFirst	True
NbPrior	True
NbNext	True
NbLast	True
Nbinsert	False
NbDelete	False
NbEdit	False
NbPost	False
NbCancel	False
NbRefresh	False

24. A continuación dentro del componente GroupBox2 coloque un componente TDBGrid (pestaña Data Controls), y modifique su propiedad DataSource al valor DM.DM.DsOrders.
25. Grabe los cambios de su aplicación y ejecútela.



Observe que al momento de navegar en los registros de clientes de forma automática se refrescan los registros de Ordenes mostrando solamente aquellos que pertenecen al cliente seleccionado.

Filtros de Información

26. Enseguida colocaremos los componentes necesarios para realizar un filtro de información sobre los registros de la tabla TbCustomer, para esto coloque dentro del componente Panel1 un componente TLabel (pestaña Standard) y modifique su propiedad Caption al valor Filtro por País.
27. Dentro del componente Panell, debajo del componente Labell, coloque un componente TDBLookupComboBoxl (pestaña Data Controls), modifique su propiedad ListSource al valor DM.DSCustomer_Orders, su propiedad ListField al valor Country y su propiedad KeyField a Country.
28. A través del Inspector de Objetos, agregue el siguiente manejador para el evento OnClick del componente DBComboBox1.

```
procedure TFrmMain.DBLookupComboBox1Click(Sender: TObject);
begin
  DM.TbCustomer.Filter := 'Country= '+ QuotedStr(DM.QryCustomer_cityCOUNTRY.AsString);
  DM.TbCustomer.Filtered := TRUE;
end;
```

29. Dentro del componente Panell, a la derecha del componente DBComboBoxl, coloque un componente TSpeedButton (pestaña Additional), modifique su propiedad Caption al valor &Limpiar Filtro y en su propiedad Glyph asocie el archivo ERASE.BMP que se ubica en el directorio:

C: \Archivos de programa\Archivos comunes\Borland Shared\Images\Buttons

30. A través del Inspector de Objetos, agregue el siguiente manejador para el evento OnClick del componente SpeedButtonl.

```
procedure TFrmMain.SpeedButton1Click(Sender: TObject);
begin
  DM.TbCustomer.Filtered := False;
  DBLookupComboBox1.KeyValue := '';
end;
```

Búsqueda de Información

31. Coloque dentro del componente Panel1 dos componentes TSpeedButton, y modifique sus propiedades como se indica en la siguiente tabla:

COMPONENTE	PROPIEDAD	VALOR
SpeedButton2	Caption	Búsqueda &Exacta
	Glyph	ZoomIn.Bmp
SpeedButton3	Caption	Búsqueda Cercana
	Glyph	Zoomout.Bmp

32. A través del Inspector de Objetos, agregue el siguiente manejador para el evento OnClick del componente SpeedButton2 (Búsqueda &Exacta).

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
vnumerocliente:integer;
vencontrado  : Boolean;
begin
  dm.TBCUSTOMER.IndexFieldNames:='custno';
  vnumerocliente:=strtoint(inputbox('Búsqueda Exacta', 'Número de Cliente:',' '));
  vencontrado:=Dm.TBCUSTOMER.findkey ([vnumerocliente]);
  if not vencontrado then
  begin
    messagedlg('cliente NO Encontrado', mtinformation, [mbok], 0);
    dm.TBCUSTOMER.First;
  end;
```

```
end;
```

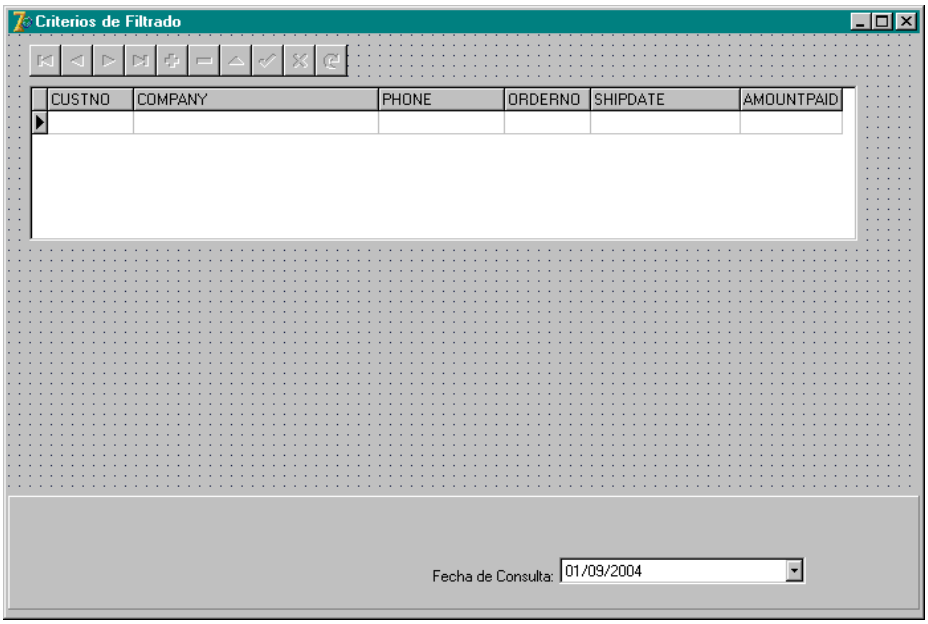
33. A través del Inspector de Objetos, agregue el siguiente manejador para el evento OnClick del componente SpeedButton3 (Búsqueda &Cercana).

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
var
 vcercano:string;
begin
  dm.TBCUSTOMER.indexfieldnames:='company';
  vcercano:=inputbox('Busqueda Cercana','Nombre de la  Compañia', ' ');
  dm.TBCUSTOMER.FindNearest([vcercano]);
end;
```

Uso del Componente Tquery

34. A continuación crearemos una segunda forma para visualizar el resultado de la consulta SQL configurada en el componente QryCustomer_Orders para esto, seleccione del menú de Delphi la opción File | New | Form y salve el archivo de la forma como UConsulta.pas.

Después de colocar algunos componentes en la nueva forma de consulta deberá lucir como la siguiente figura.



35. Modifique la propiedad Name de la nueva forma al valor FrmConsulta y su propiedad Caption a Criterios de Filtrado.
36. Del menú de Delphi seleccione la opción File | Use Unit y seleccione la unidad UDataModule dentro del cuadro de diálogo Use Unit. Pulse click en el botón OK
37. Agregue un componente TDBNavigator (pestaña Data Controls), y colóquelo en el extremo superior izquierdo de la forma FrmConsulta y modifique su propiedad DataSource al valor DM.DSCustomer_Orders2.
38. Debajo del componente DBNavigator1 coloque un componente TDBGnd (pestaña Data Controls), y modifique su propiedad Data Source al valor DM.DSCustomer_Orders2.
39. En la forma FrmConsulta coloque un componente TPanel (pestaña Standard), modifique su propiedad Align al valor alBottom y elimine el contenido de su propiedad Caption.
40. Coloque dentro del componente Panel1 un componente TLabel (pestaña Standard) y modifique su propiedad Caption al valor Fecha de Consulta.
41. Debajo del componente Label1, coloque un componente TdateTimePicker (pestaña Win32).
42. A través del Inspector de Objetos, agregue el siguiente manejador para el evento OnChange del componente DateTimePicker.

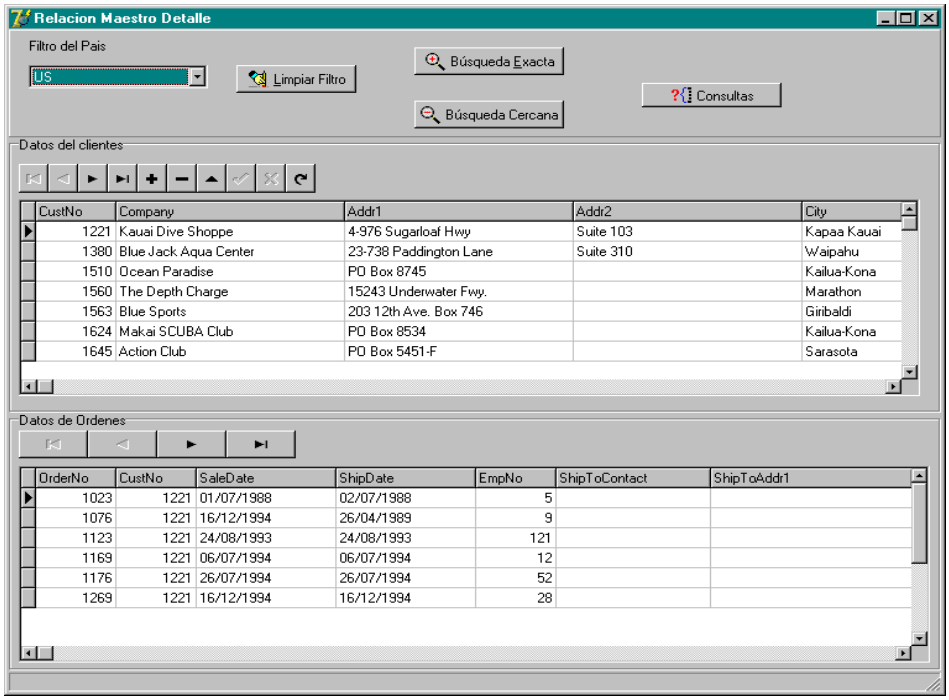
```
procedure Tfrmconsulta.DateTimePicker1Change(Sender: TObject);
begin
with DM.QRYCUSTOMER_ORDERS do
 begin
  try
   disablecontrols;
   active:=false;
   parambyname('fecha_venta').AsDate:=datetimepicker1.Date;
   active:=true;
  finally
   enablecontrols;
  end;
 end;
end;
```

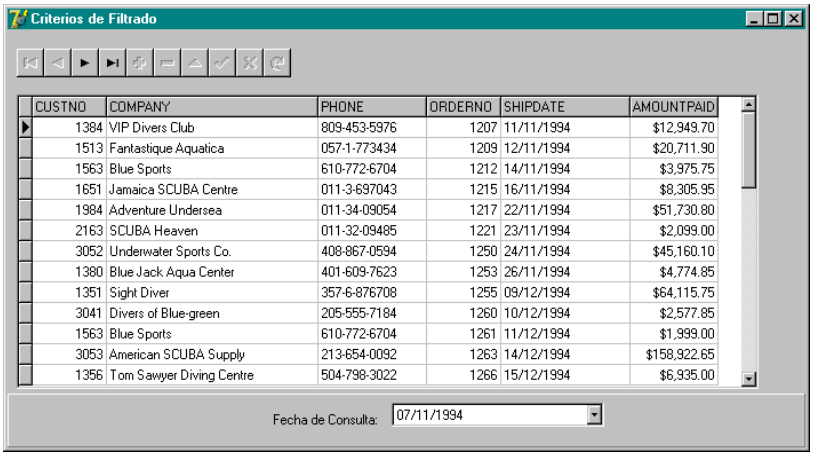
43. Volvamos a la forma principal (FrmMain), del menú de Delphi seleccione la opción File | Use Unit y seleccione la unidad UConsulta dentro del cuadro de diálogo Use Unit. Pulse click en el botón OK

44. A través del Inspector de Objetos, agregue el siguiente manejador al evento OnClick del SpeedButton4 Consultas.

```
procedure TFrmMain.SpeedButton4Click(Sender: TObject);
begin
  FrmConsulta.showModal;
end;
```

45. Salve sus cambios y ejecute la aplicación para probar su funcionalidad.





8.3 InterBase Express

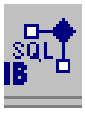
Cuando se decide que una aplicación utilizará invariablemente un servidor de base de datos dado, se puede determinar no utilizar ningún motor de base de datos o librería y escribir programas que estén directamente unidos a la API del servidor específico de bases de datos, lo cual hará sus programas intrínsecamente no portables a otros servidores de SQL.

En ocasiones podemos desarrollar aplicaciones que utilicen componentes nativos o de terceras compañías, que encapsulen las APIs del servidor de bases de datos, un ejemplo de esa familia de componentes es InterBase Express (IBX). Las aplicaciones desarrolladas utilizando estos componentes deberían de ser más rápidas, y ofrecer más control sobre las características especificas del servidor. Por ejemplo, IBX proporciona un conjunto de componentes administrativos construidos específicamente para Interbase 6.

Componentes Dataset IBX

El conjunto de componentes IBX incluye algunos componentes personalizados dataset y algunos otros. Los componentes dataset o de conjuntos de datos, pueden utilizar todos los controles comunes data-aware (Carpeta Data Controls) de Delphi, estos incluyen un editor de campos y todas las características comunes de tiempo de diseño, pero no requieren el uso del BDE.

IBX permite seleccionar entre tres componentes dataset. Estos componentes poseen características similares a sus contra partes del BDE y se describen a continuación:

COMPONENTE	DESCRIPCION
IBTable 	Similar al componente TTable permite acceder a una sola tabla o vista.
IBQuery 	Similar al componente TQuery permite ejecutar una sentencia SQL y devolver un conjunto de datos, puede utilizarse en conjunción con el componente TIBUpdateSQL para obtener un conjunto de datos editables.
IBStoredProc 	Similar al componente TStoredProc, permite ejecutar un procedimiento almacenado.

Para desarrollar aplicaciones nuevas, es preferible utilizar el componente TIBDataSet, el cual permite trabajar con un conjunto de datos editable al ejecutar una sentencia select en un query. Los tres componentes descritos anteriormente se mantienen por compatibilidad con aplicaciones que utilicen BDE.

Existen otros componentes en InterBase Exprese tales como:

COMPONENTE	DESCRIPCIÓN
TIBDatabase 	Similar al componente TDatabase y se utiliza para configurar una conexión a la base de datos.
TIBTransaction 	Permite tener control sobre las transacciones. En Interbase es importante utilizar las transacciones explícitamente y aislar cada transacción apropiadamente.
TIBSQL 	Permite ejecutar sentencias SQL que no devuelvan un conjunto de datos sin utilizar la sobrecarga de un componente dataset.
TIBDatabaseInfo 	Se utiliza para inspeccionar la estructura de la base de datos y su status.
TIBSQLMonitor 	Se utiliza para depurar las aplicaciones.
TIBEvents 	Recibe eventos publicados por el servidor.

Estos componentes proporcionan más control sobre el servidor de base de datos que el que se tiene con el BDE. Por ejemplo, el tener un componente específico para manejar transacciones permite manejar múltiples

transacciones concurrentes sobre una o múltiples bases de datos, además el componente TIBDatabase permite crear una base de datos, probar la conexión, y generalmente acceder información del sistema, algo que los componentes del BDE no proporcionan completamente.

8.4 DBExpress

DbExpress es una capa independiente de la base de datos, interplataforma que proporciona métodos para procesamiento dinámico de SQL. Define una interfaz común para acceder a una variedad de servidores SQL, para cada servidor soportado, dbExpress proporciona un driver, el cual actúa como una librería independiente que implementa las interfaces comunes de dbExpress para procesamiento de queries y procedimientos almacenados. Los drivers de dbExpress están disponibles tanto para el sistema operativo Windows como para Linux como librerías de ligado dinámico (Windows) o archivos de objetos compartidos (Linux).

DbExpress se diseñó para ser rápido, ligero y fácil de instalar. Como tal, realiza muy poco procesamiento de datos, pero principalmente actúa como una envoltura delgada alrededor del software del lado del cliente del servidor de bases de datos. Proporciona las ventajas de un API común sin la sobrecarga de un motor más elaborado de bases de datos como el Borland Database Engine. Por ejemplo, dbExpress devuelve solamente cursores unidireccionales, y no posee cache de datos o metadatos. Manteniendo la capa central de acceso a datos en tiempo de ejecución delgada y simple, dbExpress proporciona una conectividad a bases de datos que puede ser adaptada fácilmente a nuevas fuentes de datos.

En la parte superior de las interfaces centrales, los desarrolladores de aplicaciones pueden crear una nueva capa y proporcionar navegabilidad hacia atrás y hacia delante en el conjunto de datos devueltos. Por ejemplo, componentes tales como un Client Dataset pueden proporcionar el soporte para cache, navegación, indexado y filtrado de los conjuntos de datos devueltos por dbExpress.

Mapeo de Tipos de Datos

Las fuentes de datos representan los datos en varios formatos dbExpress unifica esta variedad en un conjunto de tipos de datos genéricos para obtener los valores individuales de los campos dbExpress define un conjunto de tipos de datos lógicos que asemejan a la mayoría de los tipos de datos soportados por varios tipos de bases de datos. Aunque existe una traducción entre los tipos lógicos y los de SQL, la sobrecarga es imperceptible.

La siguiente tabla muestra los tipos de datos lógicos y subtipos utilizados por dbExpress. Internamente dbExpress mapea los tipos SQL específicos de la base de datos en un tipo de dato lógico. dbExpress no expone estos tipos de datos físicos a los clientes.

Tipo de Campo(lógico)	Subtipo de Campo	Significado
FldUNKNOWN		Tipo de campo desconocido
FldZSTRING		Cadena terminada en nulo
FldZSTRING	FldstFIXED	CHAR
FidDATE		Fecha (32 bits)
FidDATE	FldstADTDATE	Fecha en un ADT
FidBLOB		Objeto largo binario (BLOB)
FidBLOB	FldstMEMO	Texto Memo
FldBLOB	FldstBINARY	Datos binarios
FidBLOB	FldstHMEMO	BLOB
FldBLOB	FldstHBINARY	BLOB
FldBLOB	FldstBFlLE	BFILE
FidBOOL		Booleano (16 bits)
FldlNT16		Entero con signo 16-bits
FldlNT32		Entero con signo 32-bits
FldlNT32	FldstAUTOINC	Campo auto-incremental
FldFLOAT		Punto flotante de 64-bits
FldFLOAT	FldstMONEY	Moneda
FidBCD		Decimal Código-Binario (BCD)
FidBYTES		Número fijo de bytes
FldTIME		Hora (32 bits)
FidVARBYTES		Variable de bytes de longitud prefijada
FldDATETIME		Campo de estructura DateTime
FldCURSOR		Parámetro de cursor utilizado por los procedimientos almacenados que devuelven un cursor como parámetro
FidADT		Tipo de dato abstracto (estructura), los cursores utilizan fIdADT para indicar el número de miembros contenidos en el campo ADT.
FidADT	FldstADTNestedTable	Tabla anidada
FidARRAY		Campo de tipo de arreglo
FldREF		Referencia a ADT (actualmente no soportado)

FidTABLE		Tabla anidada (referencia) (actualmente no soportado)
FldFMTBCD		Decimal de Código-Binario (BCD)

Estos tipos de datos lógico, se asemejan a los tipos de datos lógicos utilizados por el Borland Database Engine, lo cual permite la compatibilidad y la portabilidad más fácil de aplicaciones. Sin embargo, el mapeo desde SQL hasta los tipos de datos lógicos puede no ser el mismo que en BDE. Por ejemplo, fIdDATETIME es introducido para representar datos timestamp sin pérdida de datos, similarmente, los datos numéricos que no pueden caber en un double son mapeados directamente a un decimal de código-binario (BCD).

8.5 Preguntas y Ejercicios de Repaso

1. Describa las características de IBExpress.

2. Menciona 3 componentes de BDE.

3. Subraya la herramienta que se encarga de encapsular una tabla de una base de datos

a)Ttable b)Tquery c)Tdatabase

4. Describa las características del BDE.

5. Crea una base de datos con los datos necesarios para una agenda.

6. Utilizando la base de datos anterior crea una forma de captura y de consulta donde filtres por ciudad.

7. Describa las características de DBExpress.

8. ¿Cuáles son las diferencias entre BDE y DBEXpress?

9. Crea un programa que pueda obtener las ventas de una tienda por dia y que tambien me sirva para consultar los precios de ventas.

Ejercicio de Repaso

Conexiones DBE

1. Cree una nueva aplicación File | New | Application, salve su proyecto con los siguientes nombres de archivo:

Archivo de la forma: UMain.pas

Archivo del proyecto: Scomunitarios.dpr

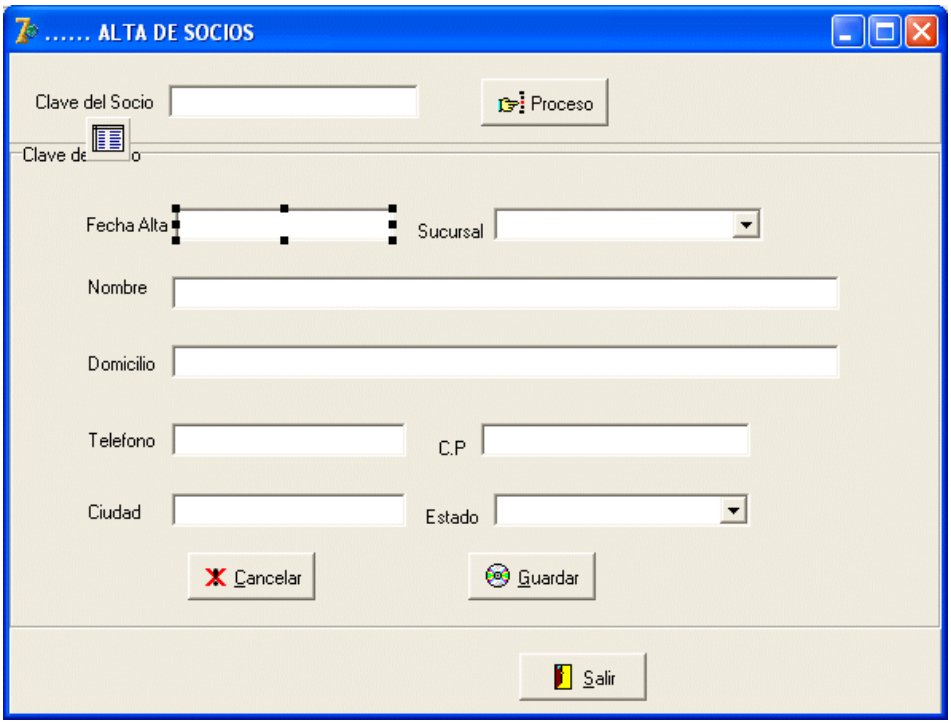
2. A continuación agregamos dos panel (pestaña standard)
3. Agregamos 9 Tlabel, 7 Tedit y 2 TComboBox (pestaña standard)
4. Agregamos 4 TBitbtn (pestaña Additional)
5. Un componente Ttable sobre la forma en esta ocasión no utilizaremos DataModule
6. Modifique las propiedades de dichos componentes como se muestra a continuación:

En la propiedad DatabaseName de Table1escribiremos el alias donde esta almacenada la base de datos, asi relacionamos la base de datos con nuestra aplicación.

COMPONENTE	PROPIEDAD	VALOR
Table1	Name	Socios
	DatabaseName	Comunitar
	TableName	Clientes
	Active	True
Tlabel1	Caption	Clave del socio
Tlabel2	Caption	Fecha Alta
Tlabel3	Caption	Sucursal
Tlabel4	Caption	Nombre
Tlabel5	Caption	Domicilio
Tlabel6	Caption	Telefono
Tlabel7	Caption	C.P
Tlabel8	Caption	Ciudad
Tlabel9	Caption	Estado
Tcombobox1	Items	CENTRO SUR NORTE SUROESTE ESTE NOROESTE
Tcombobox2	Items	Escribiremos Estados de la Republica

TbitBtn1	Name Caption Glyph	BitBtn1 Proceso ARROW3R.bmp
TbitBtn2	Name Caption Glyph	BitBtn2 Cancelar Ignor.bmp
TbitBtn3	Name Caption Glyph	BitBtn3 Guardar cd.bmp
TbitBtn4	Name Caption Glyph	BitBtn4 Salir Close.bmp

7. En todos los Tedit tenemos que quitarle todo lo que viene en Text.
8. Ordenalos para que tenga este aspecto.



9. Hacer doble click sobre el botón Proceso para generar su evento y escribimos lo siguiente.

```
procedure Talta_cli.BitBtn1Click(Sender: TObject);
var
clientetmp:string;
begin
IF EDIT1.Text<>'' THEN
BEGIN
 Edit8.text:='   '+ datetostr(date);
 Edit8.readonly:=true;
 table1.Active:= true;
 table1.Open;
table1.First;
clientetmp:= edit1.Text;
  if table1.Locate('clave', clientetmp,[locaseinsensitive]) then
  begin
    showmessage('Este cliente ya Existe');
    edit1.Text:='';
  end
    else
    begin
        BitBtn4.Enabled:=FALSE;
        panel1.Enabled:= false;
       groupbox1.Visible:= true;
       EDIT2.SetFocus;
    end;
END
ELSE
begin
  SHOWMESSAGE('Intoduzca un Número Valido');
  edit1.SetFocus;
end;
end;
```

Este proceso busca el número de socio en la base de datos para verificar si no esta ya dado de alta.

10. Hacer doble click sobre el botón Cancelar para generar su evento y escribimos lo siguiente.

```
procedure Talta_cli.BitBtn2Click(Sender: TObject);
begin
 EDIT2.Text:='';
```

```
  EDIT3.Text:='';
  EDIT4.Text:='';
  EDIT5.Text:='';
  EDIT6.Text:='';
  EDIT7.Text:='';
  edit9.Text:='';
  GroupBox1.Visible:=FALSE;
  PANEL1.Enabled:=TRUE;
  EDIT1.Text:='';
  BitBtn4.Enabled:=TRUE;
  EDIT1.SetFocus;
end;
```

11. Hacer doble click sobre el botón Guardar para generar su evento y escribimos lo siguiente.

```
procedure Talta_cli.BitBtn3Click(Sender: TObject);
begin
  TABLE1.Append;
  TABLE1.FieldByName('CLAVE').AsString:=EDIT1.Text;
  TABLE1.FieldByName('NOMBRE').AsString:=EDIT2.Text;
  TABLE1.FieldByName('DOMICILIO').AsString:=EDIT3.Text;
  TABLE1.FieldByName('TELEFONO').AsString:=EDIT4.Text;
  TABLE1.FieldByName('CP').AsCurrency:=STRTOINT(EDIT5.Text);
  TABLE1.FieldByName('CIUDAD').AsString:=EDIT6.Text;
  TABLE1.FieldByName('ESTADO').AsString:=EDIT7.Text;
  TABLE1.FieldByName('FECHA').AsString:=EDIT8.Text;
  table1.FieldByName('SUCURSAL').AsString:=EDIT9.Text;
  table1.FieldByName('ACTIVO').AsString:='ACTIVO';
  TABLE1.Post;
  EDIT1.Text:='';
  EDIT2.Text:='';
  EDIT3.Text:='';
  EDIT4.Text:='';
  EDIT5.Text:='';
  EDIT6.Text:='';
  EDIT7.Text:='';
  EDIT9.Text:='';
```

```
  GroupBox1.Visible:=FALSE;
  PANEL1.Enabled:=TRUE;
  EDIT1.SetFocus;
  BitBtn4.Enabled:=TRUE;
end;
```

12. Hacer doble click sobre el botón Salir para generar su evento y escribimos lo siguiente.

```
procedure Talta_cli.BitBtn4Click(Sender: TObject);
begin
 EDIT1.Text:='';
 CLOSE;
end;
```

13. Salve sus cambios y ejecute la aplicación para probar su funcionalidad.

XI. Desarrollo de Aplicaciones con dbExpress.

9.1 Objetos de Conexión de dbExpress

Los componentes de acceso a datos de dbExpress se encuentran en la pestaña dbExpress de la paleta de componentes.

9.1.1 Componente TSQLConnection

"TSQLConnection. Encapsula una conexión de dbExpress al servidor de base de datos.

La conexión proporcionada por un solo componente TSQLConnection puede compartirse entre múltiples componentes datasets de SQL a través de su propiedad SQLConnection.

Este componente interactúa con un driver de dbExpress y dos archivos. El primero dbxdrivers.ini en Windows (dbxdrivers en Linux), lista tos tipos de drivers instalados, y para cada driver, lista las librerías (DLLs u objetos compartidos) que requiere así como también la configuración por defecto de todos los parámetros de conexión. El segundo archivo, dbxconnections.ini en Windows (dbxconnections en Linux), lista conjuntos de parámetros configuración de conexiones identificados por un nombre. Cada configuración representa un conjunto de valores de TSQLConnection y describe una conexión particular de base de datos.

Se puede utilizar TSQLConnection para escoger entre una de las configuraciones nombradas en dbxconnections.ini y utilizarla para conectarse a una base de datos o definir nuevas configuraciones y agregarlas a dbxconnections.ini."[1]

En la siguiente tabla se describen las propiedades del componente TSQLConnection.

PROPIEDAD	DESCRIPCIÓN
ActiveStatements	Indica en número de declaraciones actualmente activas en el servidor de bases de datos.
AutoClone	Especifica si la conexión de SQL automáticamente clona las conexiones de base de datos cuando se necesitan.
Connected	Especifica si la conexión está activa.
ConnectionName	Nombra la configuración de conexión.
ConnectionState	Indica el estado actual del objeto TSQLConnection
DataSets	Lista todos los datasets activos para el componente de conexión.
DriverName	Especifica el driver de base de datos asociado con la conexión de SQL.
GetDriverFunc	Especifica la función que devuelve el driver de alto nivel de dbExpress.

[1] http://www.programacion.net/foros/28/msg/141755 Online+Pdf%20Docs.zip

InTransaction	Indica si una transacción está en progreso.
KeepConnection	Especifica si la conexión debe permanecer activa cuando no existen dataseis abiertos.
LibraryName	Especifica el driver de dbExpress que se utiliza para establecer una conexión

PROPIEDAD	DESCRIPCIÓN
LoadParamsOnConnect	Especifica si el componente TSQLConnection carga la configuración nombrada por ConnectionName inmediatamente antes de conectarse al servidor.
LocalCode	Identifica el almacén que es utilizado para elementos de los datasets que utilizan esta conexión.
MaxStmtsPerConn	Indica el límite del servidor en el número de sentencias soportadas por una sola conexión de base de base de datos.
MetaData	Proporciona una interfaz al objeto de dbExpress que proporciona los metadatos del servidor.
MultipleTransactionsSupported	Indica si el servidor de bases de datos soporta múltiples transacciones.
Parama	Lista los parámetros de la conexión.
ParamsLoaded	Indica cuando los valores de los parámetros han sido cargados desde dbxconnections.ini.
SQLConnection	Proporciona una interfaz al objeto de dbExpress que representa una conexión.
SQLHourGlass	Especifica cuando debe de cambiar el cursor a crSQLWait durante operaciones largas.
TableScope	Indica que tipos de tablas son devueltas cuando se recupera información del esquema de las tablas.
TraceCallbackEvent	Proporciona acceso a la función callback que se ejecuta para cada comando de SQL pasado desde o hacia el servidor.
TransactionsSupported	Indica si el servidor de bases de datos soporta transacciones.
VendorLib	Especifica la librería de cliente (DLL u objeto compartido) proporcionada por el vendedor de base de datos.

9.1.2 Componente TSQLDataSet

TSQLDataSet

Representa los datos devueltos utilizando dbExpress.

TSQLDataSet representa un conjunto de datos unidireccional para acceder la información de la base de datos utilizando dbExpress.

Este componente se puede utilizar para:

Representar los registros de una tabla de una base de datos, el resultado de un query, o el resultado devuelto por un procedimiento almacenado.

Ejecuta una consulta SQL (Query) o un procedimiento almacenado que no devuelva un conjunto de datos.

Representar metadatos que describan que está disponible en e! servidor de bases de datos (tablas, procedimientos almacenados, campos en una tabla, etc.).

TSQLDataSet es un conjunto de datos (dataset) unidireccional. A diferencia de otros datasets, los datasets unidireccionales no almacenan múltiples registros en memoria. Debido a esto, solo se pueden navegar utilizando los métodos First y Next. No existe soporte predefinido para la edición de datos, se debe de crear una sentencia de SQL UPDATE o se puede conectar el dataset a un cfient dataset utilizando un provider.

Características que requieren almacenar múltiples registros, tales como filtrado o campos de búsqueda no están disponibles.

La siguiente tabla lista las propiedades del componente TSQLDataSet.

PROPIEDAD	DESCRIPCIÓN
CommandText	Especifica el comando que ejecuta el dataset.
CommandType	Indica el significado de la propiedad CommandText.
DataSource	Liga el dataset de SQL a otro dataset (maestro).
DesignerData	Almacena datos personalizados
IndexDefs	Contiene las definiciones de todos los índices definidos en el servidor para el dataset.
MaxBlobSize	Indica el número máximo de bytes recuperados para cualquier campo BLOB en el dataset.
NoMetaData	Especifica si el dataset de SQL recupera información de metadatos junto con los datos (deprecado)
ParamCheck	Especifica si la lista de parámetros de un dataset de SQL se regenera cuando el comando de SQL se modifica.
Params	Representa los parámetros de un query o un procedimiento almacenado.

PROPIEDAD	DESCRIPCIÓN
Prepared	Especifica si el comando se prepara antes de su ejecución
RecordCount	Indica el número total de registros asociados con el dataset.
SortFieldNames	Indica el tiempo de ordenamiento cuando CommandType es clTable.
SQLConnection	Especifica la conexión de SQL que conecta el dataset a un servidor de bases de datos.
TransactionLevel	Indica la transacción a la cual pertenece el dataset.

La tabla siguiente lista los métodos del componente TSQLDataSet.

MÉTODO	DESCRIPCIÓN
Create	Crea e instancia un componente TSQLDataSet
ExecSQL	Ejecuta un query o un procedimiento almacenado que no devuelve un conjunto de datos.

9.1.3 Componente TSQLQuery

TSQLQuery

Representa un query que es ejecutado utilizando dbExpress.

TSQLQuery puede representar el resultado de una sentencia SELECT o realizar acciones en el servidor de bases de datos utilizando sentencias como INSERT, DELETE, UPDATE, ALTER TABLE, etc. Representa un dataset unidireccional.

En la siguiente tabla se listan las propiedades del componente TSQLQuery.

PROPIEDAD	DESCRIPCIÓN
SQL	Especifica el comando de SQL a ejecutar en el servidor de base de datos.
SQLConnection	Especifica el componente de conexión SQL que conecta el dataset al servidor de base de datos
Text	Indica la sentencia SQL en una cadena.

A continuación se listan los métodos definidos por el componente TSQLQuery.

MÉTODO	DESCRIPCIÓN
Create	Crea e inicializa un componente TSQLQuery.
Destroy	Destruye la instancia de TSQLQuery.
ExecSQL	Ejecuta un query que no devuelve un conjunto de datos.
PrepareStatement	Prepara el query para su ejecución.

9.1.4 Componente TSQLStoredProc

TSQLStoredProc

Representa un procedimiento almacenado que se ejecuta utilizando dbExpress.

TSQLStoredProc puede representar el conjunto de datos unidireccionales devueltos por el procedimiento almacenado.

La tabla siguiente lista las propiedades definidas por el componente TSQLStoredProc.

MÉTODO	DESCRIPCIÓN
PackageName	Especifica el nombre del Paquete de Oracle al cual pertenece el procedimiento almacenado.
SQLConnection	Especifica el componente de conexión SQL que conecta el dataset al servidor de base de datos
StoredProcName	Especifica el nombre del procedimiento almacenado.

La tabla siguiente lista los métodos definidos por el componente TSQLStoredProc.

MÉTODO	DESCRIPCIÓN
Create	Crea e instancia un componente TSQLStoredProc.
ExecProc	Ejecuta el procedimiento almacenado cuando este no devuelve un cursor.
NextRecordSet	Proporciona acceso a un conjunto de registros secundario.
PrepareStatement	Prepara el procedimiento almacenado para su ejecución.

9.1.5 Componente TSQLTable

TSQLTable.

Representa una tabla de la basa de datos que se accede a través de dbExpress. Este componente genera un query para recuperar todos los registros y columnas de la tabla especificada. TSQLTable es un dataset unidireccional.

A continuación se listan las principales propiedades definidas por el componente TSQLTable.

PROPIEDAD	DESCRIPCIÓN
IndexFieldCount	Indica el número total de campos que forman el índice actual.
IndexFieldNames	Lista los campos que se utilizan para ordenar los registros y para ligado en relaciones maestro/detalle.
IndexFields	Especifica los campos asociados con el índice actual.
IndexName	Identifica un índice a utilizar para ordenar registros y para ligar una tabla como el detalle en una relación maestro/detalle.
MasterFields	Identifica los campos del dataset maestro que se utilizan para ligar esta tabla como el detalle en una relación maestro/detalle.
MasterSource	Identifica el dataset maestro a utilizar cuando se liga esta tabla como el detalle en una relación maestro/detalle.
SQLConnection	Especifica el componente de conexión SQL que conecta el dataset al servidor de base de datos
TableName	Especifica la tabla del servidor de base de datos representada por este componente.

La siguiente tabla lista los métodos definidos por el componente TSQLTable.

MÉTODO	DESCRIPCIÓN
Create	Crea e instancia un componente TSQLTable.
DeleteRecords	Vacia la tabla asociada
Destroy	Destruye la instancia de TSQLTable.
GetIndexNames	Devuelve una lista de los índices disponibles definidos para la tabla.
PrepareStatements	Genera el query que el componente SQLTable utiliza para recuperar los datos del servidor de base de datos.

9.1.6 Componente TSQLMonitor.

TSQLMonitor

Intercepta los mensajes que se pasan entre un componente de conexión SQL y un servidor de base de datos y los salva en una lista de cadenas.

TSQLMonitor se utiliza para depurar la comunicación entre una aplicación y un servidor de base de datos. Cada instancia de TSQLMonitor guarda en un log los comandos de SQL de un componente de conexión SQL en particular, agregándolos a una lista de cadenas. Esto permite no solamente ver los comandos que explícitamente agregamos a un dataset de SQL, sino también comandos que se generan en el servidor.

La siguiente tabla lista las propiedades definidas por el componente TSQLMonitor.

PROPIEDAD	DESCRIPCIÓN
Active	Inicia o detiene el monitor
AutoSave	Especifica si los mensajes monitoreados se salvan automáticamente a un archivo.
FileName	Especifica el archivo donde se almacenan los mensajes monitoreados.
MaxTraceCount	Indica el número máximo de mensajes que pueden ser salvados.
SQLConnection	Especifica el componente de conexión cuyos mensajes se van a monitorear.

TraceCount	Indica el número de mensajes actualmente salvados.
TraceList	Lista los mensajes que han sido pasados entre el SQLConnection y el servidor de base de datos.

La siguiente tabla lista los métodos definidos por el componente TSQLMonitor.

MÉTODO	DESCRIPCIÓN
Create	Crea e instancia un componente TSQLMonitor
Destroy	Destruye el objeto TSQLMonitor.
LoadFromFile	Coloca el contenido de TraceList desde las cadenas almacenadas en un archivo.
SaveToFile	Salva el contenido de TraceList a un archivo.

9.1.7 Componente TSimpleDataSet.

TSimpleDataSet

Utiliza dbExpress para recuperar los datos y almacenarlos en un cache interno en memoria.

TSimpleDataSet es un Client Dataset que utiliza un TSQLDataSet interno y un TDataSetProvider para recuperar datos y aplicar actualizaciones. Combina el acceso rápido y la facilidad de uso de un dataset unidireccional junto con la habilidad de un client dataset para editar y navegar por los datos.

TSimpleDataSet utiliza dbExpress para obtener un acceso rápido a la información de la base de datos. Almacena la información en memoria y salva las actualizaciones con su dataset provider interno. Por lo tanto actúa como un TClientDataset cuando está conectado a un TSQLDataSet vía un provider, excepto que el dataset fuente y el provider son internos.

La tabla siguiente lista las propiedades del componente TSimpleDataSet.

PROPIEDAD	DESCRIPCIÓN
Connection	Identifica el componente TSQLConnection que conecta el dataset al servidor de base de datos.
DataSet	Proporciona acceso al dataset interno que recupera los datos del servidor

La siguiente tabla lista los métodos definidos en el componente TSimpleDataSet.

MÉTODO	DESCRIPCIÓN
Create	Crea e instancia un objeto TSimpleDataSet
Destroy	Destruye la instancia de TSimpleDataSet

El componente TSimpleDataSet define un único evento OnReconcileError el cual ocurre cuando un client dataset necesita reconciliar una actualización a un registro que no pudo aplicarse.

La siguiente tabla compara al Borland Database Engine y a dbExpress en varias áreas mostrando sus diferencias.

CARACTERISTICA	BDE	DBEXPRESS
Almacenamiento temporal de registros	BDE determina cuantos registros se almacenan en la memoria	El programador determina cuantos registros se almacenan en memoria
Control de tráfico de Red	BDE determina cuantos registros son recuperados desde el servidor	El programador determina cuantos registros son recuperados desde el servidor y cuando.
Control de Transacciones	Están disponibles tanto un control manual como automático de transacciones. Las transacciones están activas mientras el usuario está editando información.	Están disponibles tanto un control manual como automático de transacciones. Las transacciones están activas brevemente mientras las actualizaciones están siendo aplicadas.

Instalación	La instalación ocupa aproximadamente 18 megabytes. Tanto la instalación como la configuración son complejas y requieren cambios al registro.	La instalación requiere dos DLLs (menos de medio megabyte), las cuales se pueden compilar dentro del ejecutable, no requiere cambios al registro.
Interplataforma	Windows solamente	Windows y Linux
Drivers de terceros	Difíciles de desarrollar, pocos en existencia.	Fáciles de desarrollar. Muchos disponibles.

La siguiente tabla muestra los componentes equivalentes de dbExpress para cada componente de BDE.

BDE	**DbExpress**
TdataBase	TSQLConnection
Tquery	TSQLQuery
TstoredProc	TSQLStoredProc
Ttable	TSQLTable
NA	TSQLDataSet
TbatchMove	NA
Utilieria SQL Monitor	TSQLMonitor
Session	NA
UpdateSQL	NA
NestedDataSet	NA

9.2 Preguntas de Repaso

1. Describa las características de dbExpress.

2. ¿Cuál es la función de los archivos dbxconnection.ini y dbxdrivers.ini?

3. ¿Qué es un dataset unidireccional?

4. ¿Cuál es la función y para que me sirve el componente TSQLConnection

5. Describa la función del componente TsimpleDataSet.

6. Cree un programa donde capture las ventas a credito y que me muestre los abonos de cada cliente.

7. Relaciona la descripción que corresponde a cada propiedad

Connected	Lista todos los datasets activos para el componente de conexión.
ConnectionName	Especifica el driver de base de datos asociado con la conexión de SQL.
ConnectionState	Nombra la configuración de conexión.
TSQLConnection DataSets	Especifica la función que devuelve el driver de alto nivel de dbExpress.
DriverName	Indica el estado actual del objeto.
GetDriverFunc	Especifica si la conexión está activa.

8. Crea un programa donde utilices la propiedad maestro-detalle.

X. Manejo de Excepciones y Depuración.

10.1 Introducción a Manejo de Excepciones en Delphi

"Delphi proporciona soporte para manejo de excepciones, esto es, permite desarrollar programas más robustos al tener la capacidad de manejar errores, tanto de software como de hardware de manera uniforme, de tal forma que al encontrar un error un programa sea capaz de recuperarse de este o de terminar de una forma limpia, tal vez permitiendo al usuario guardar información antes de finalizar.

La filosofía detrás del manejo de excepciones consiste en separar la lógica de una aplicación del manejo de errores, logrando de esta forma generar un código más limpio, legible y fácil de modificar, por ejemplo en lugar de generar un cuadro de diálogo reportando un error, se podría guardar un registro en un archivo de log.

La sintaxis para el manejo de excepciones es:

```
try
  Sentencias
except
BloqueDeManejoExcepciones
end;
```

Donde Sentencias es una secuencia de sentencias (separadas por ';') y BloquedeManejoExcepciones puede ser:

- Una Secuencia de sentencias
- Una Secuencia de Manejadores de Excepciones, opcionalmente seguidas por una sección else.

Un Manejador de Excepción tiene la forma:

```
on identificador.tipo do sentencias
```

Donde identificador es opcional (si se incluye puede ser cualquier tipo válido de identificador), tipo es un tipo de excepción y sentencias es cualquier bloque de sentencias.

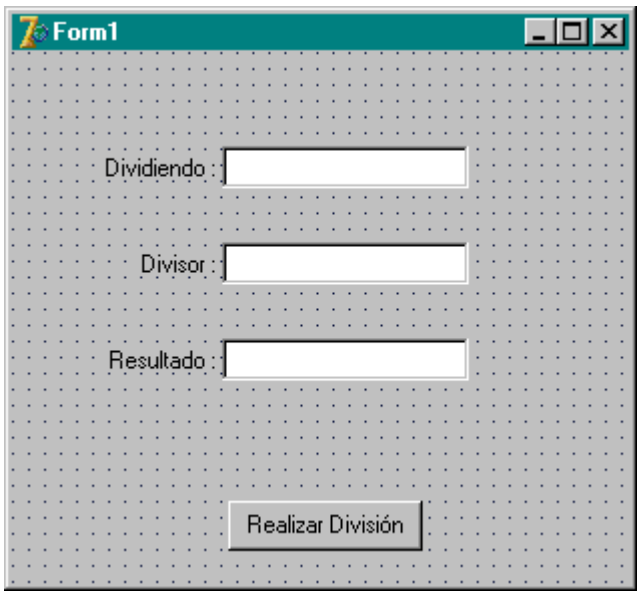
Un bloque try..except ejecuta las sentencias en el bloque Sentencias, si no se genera ninguna excepción, el bloque BloqueDeManejoExcepciones es ignorado y el control pasa al siguiente paso del programa.

Si se genera una excepción, durante la ejecución del bloque Sentencias se intenta manejar la excepción de la siguiente forma:
Si cualquiera de los manejadores de excepciones es del tipo (o de un tipo ancestro) de la excepción, el control pasa a dicho manejador.
Si ningún manejador de excepciones coincide con el tipo de la excepción, el control pasa a la cláusula else, si es que existe.
Si el bloque BloqueDeManejoExcepciones consta de una secuencia de sentencias sin ningún manejador de excepciones, entones el control pasa a la primera línea de la secuencia de sentencias.

Si ninguna de estas condiciones se satisface, la búsqueda continua en el BloqueDeManejoExcepciones del bloque try..except mas reciente que todavía no se haya cerrado, esto continua de forma recursiva. Si se alcanza el bloque try..except más externo y la excepción no es manejada, esta pasa al manejador de excepciones por defecto de Delphi.

Cuando una excepción es manejada se busca en la pila de llamadas el procedimiento o función que contienen el bloque try..except donde se maneja la excepción y el control se transfiere al bloque BloqueDeManejoExcepciones. Después el objeto de excepción es automáticamente destruido a través de una llamada a su destructor (Destroy) y el control se transfiere a la siguiente sentencia después del bloque try..except."[1]

Aplicación de Ejemplo



[1] IDEM

A continuación demostraremos el manejo de excepciones en Delphi, para esto:

1. Genere una nueva aplicación mediante el menú File | New | Application.
2. Agregue dentro de la forma tres componentes TLabeledEdit (pestaña Additional) y modifique sus propiedades de acuerdo a la tabla:

COMPONENTE	PROPIEDAD	VALOR
LabeledEdit1	(+)EditLabel Caption LabelPosition	 Dividendo : IpLeft
LabeledEdit2	(+)EditLabel Caption LabelPosition	 Divisor: IpLeft
LabeledEdit3	(+)EditLabel Caption LabelPosition	 Resultado : IpLeft

3. Agregue dentro de la forma un componente TButton (pestaña Standar) y modifique su propiedad Caption al valor Realizar División.
4. Salve su proyecto con los siguientes nombre de archivo:

Archivo de la forma: UMain.pas

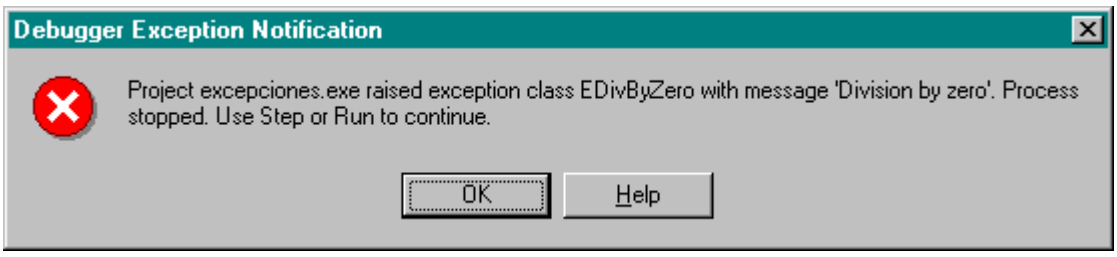
Archivo del proyecto: Excepciones.dpr

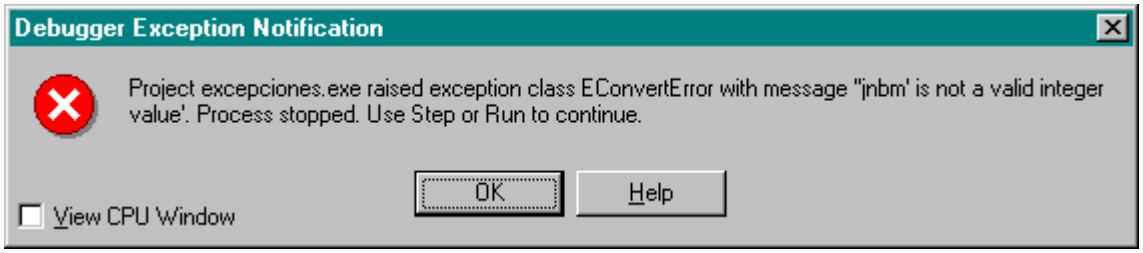
5. Agregue el siguiente manejador para el evento OnClick del Button1.

```
procedure TForm1.Button1Click(Sender: TObject);
var
resultado:integer;
begin
try
 resultado:=strtoint(labelededit1.text)div strtoint(labelededit2.text);
 labelededit3.Text:=inttostr(resultado);
except
  on edivbyzero do
    begin
```

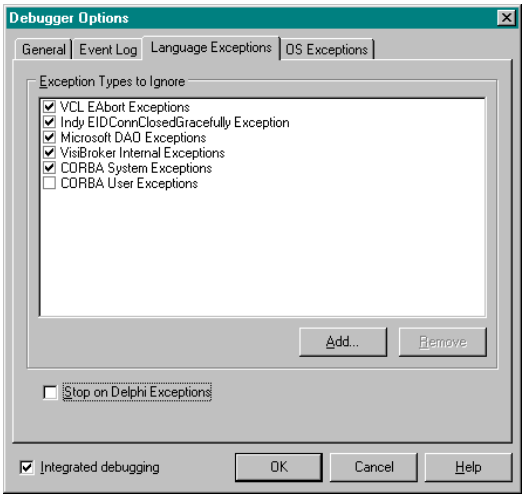
```
    showmessage('No puedo dividir entre cero');
    labelededit2.setfocus;
   end;
 on econverterror do
   showmessage('Introduzca únicamente número');
   end;
end;
```

6. Ejecute su proyecto y provoque los errores introduciendo una división entre cero e introduciendo valores que no sean numéricos.
7. Delphi deberá presentar los siguientes cuadros de diálogo

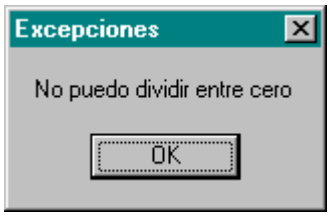


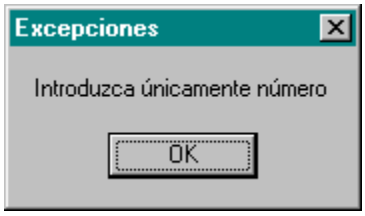


8. Pruebe nuevamente su aplicación desactivando la opción Stop on Delphi Exceptions que se encuentra en el menú de Delphi Tools | Debugger Options | Language Exceptions.



9. Ahora el manejado de excepciones deberá mostrar los siguientes mensajes de notificación al ocurrir los errores correspondientes.





10.2 El Depurador Integrado

Delphi incluye un depurador integrado disponible cuando se trabaja con el IDE.

El depurador integrado de Delphi proporciona una gran variedad de herramientas para encontrar y corregir errores en una aplicación,

Podemos clasificar los errores en tres tipos básicamente:

- Errores de Sintaxis
- Errores de Ejecución
- Errores de Lógica

Errores de Sintaxis

Los errores de Sintaxis son provocados al introducir sentencias de código erróneas como la falta de un punto y coma, no cerrar un paréntesis o una comilla, palabras reservadas mal escritas, problemas entre los tipos de datos, etc. Estos errores también se conocen como errores de compilación. Para evitarlos la clave es tener un buen conocimiento del lenguaje que se está utilizando.

Además, este tipo de errores no permiten continuar con la compilación de la aplicación, de tal manera que no podemos pasar a la fase de prueba mientras no se hayan corregido todos los errores de sintaxis.

Cuando existe algún error el usuario es notificado mediante la ventana Messages, que se ubica en la parte inferior de la unidad de Código.

Errores de Ejecución

Estos errores son inevitables, ya que se producen por ejecuciones fallidas de funciones, procedimientos y métodos, los cuales son operados por medio de la librería en Tiempo de Ejecución (Run-Time Library, RTL), la librería Visual de Componentes (Visual Component Library, VCL) o por el Sistema Operativo.

Algunos de los errores que se presentan más a menudo son: divisiones entre cero, tratar de abrir un archivo que no existe, capturar una fecha inválida, introducir números en lugar de letras o viceversa, etc.

Este tipo de errores deben tener un manejo especial y para ello Delphi introduce un concepto llamado Excepción del que se hablará más adelante.

Errores de Lógica

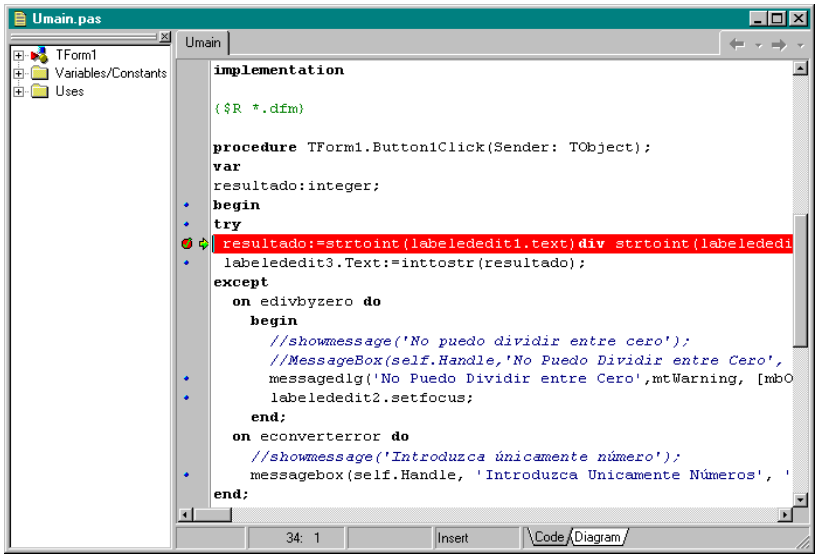
Los errores de lógica son errores en el diseño e implementación del programa. El programa contiene instrucciones válidas pero ejecuta acciones que no se tenían en mente al escribir el programa, por ejemplo, valores incorrectos en variables. Estos errores son los más difíciles de encontrar y corregir.

Además de auxiliarnos en la detección y corrección de errores, el depurador de Delphi nos permite interceptar excepciones, activar puntos de detención (breakpoint), evaluar expresiones, inspeccionar el código fuente compilado además de muchas otras funciones.

A continuación se describen las características del depurador:

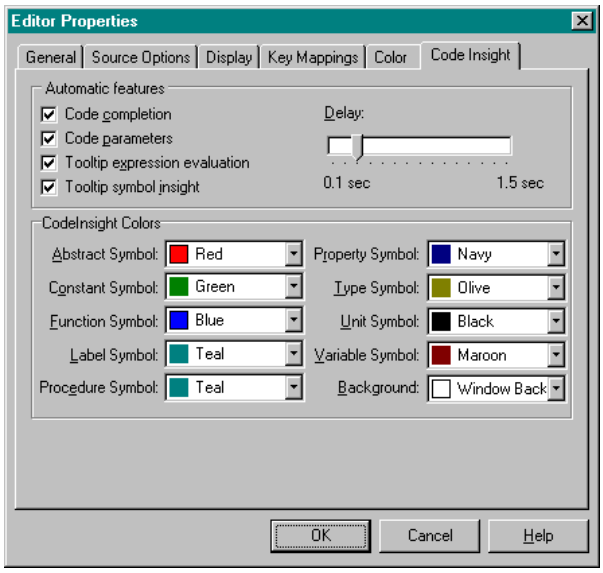
10.2.1 Evaluación de Expresiones de Tooltip

La evaluación de expresiones de Tooltip es una de las características de CodeInsight que permite evaluar fácilmente el valor de una expresión cuando el depurador está cargado en tiempo de ejecución, al posicionar el puntero del mouse sobre la expresión aparecerá una ventana desplegando el valor de la expresión, tal como se muestra en la siguiente figura.



En algunas ocasiones el valor de la expresión seleccionada no puede ser mostrado debido a optimizaciones del compilador.

La evaluación de expresiones de Tooltip puede habilitarse o deshabitarse a través del menú de Delphi Tools | Editor Options como muestra en la siguiente figura.



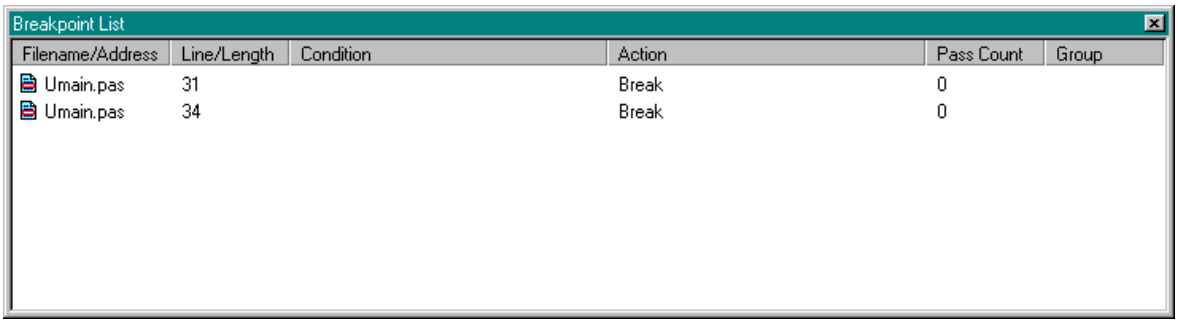
10.2.2 Ventanas de Depuración

Las ventanas de depuración, las cuales se pueden invocar a través del menú de Delphi View | Debug Windows, presentan información detallada mientras el depurador está activo, a continuación listamos las ventanas de depuración.

- **Breakpoints**
- **Call Stack**
- **Watches**
- **Local Variables**
- **Threads**
- **Modules**
- **Event Log**
- **CPU**
- **FPU**

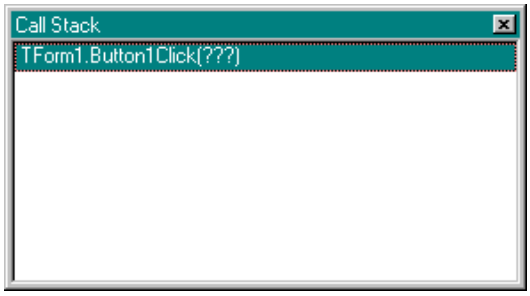
La ventana de Puntos de Detención (Breakpoints)

Esta ventana permite añadir, configurar y eliminar puntos de detención (breakpoints), los cuales se analizarán más adelante en este capítulo.



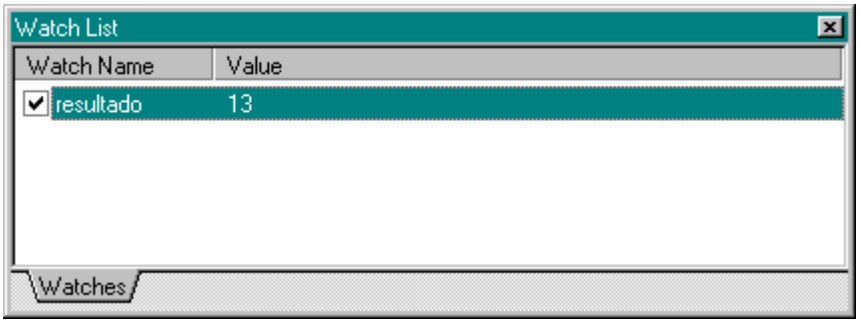
La ventana Call Stack

Esta ventana permite inspeccionar la cadena de invocaciones de subrutinas que resultaron en la ejecución de la línea de código actual en el depurador. Normalmente esta ventana solamente muestra aquellas rutinas definidas dentro de la aplicación actual para las cuales se encuentre información de símbolos de depuración.

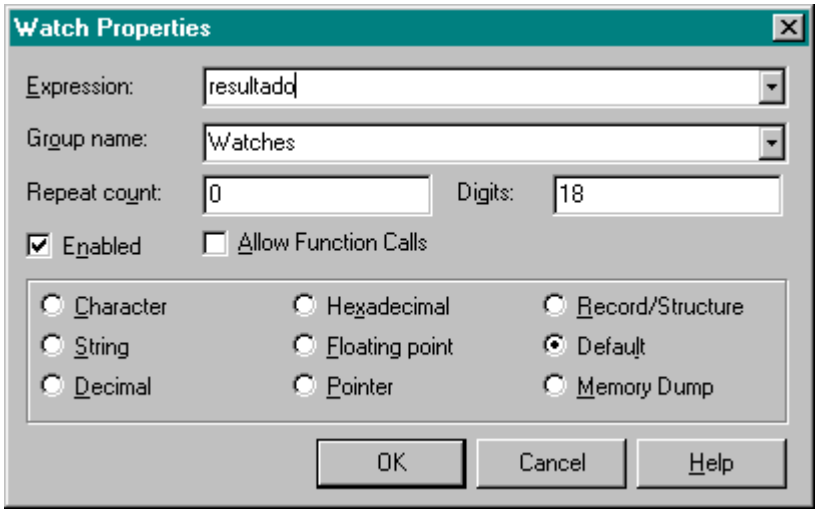


La ventana Watches

Esta ventana permite definir expresiones cuyos valores serán mostrados siempre y cuando éstas se encuentren dentro del alcance de la línea de código actual en el depurador. La siguiente figura muestra la ventana de Watches.



Para agregar una expresión a la ventana de Watches. se puede seleccionar una expresión en el editor de código y presionar las teclas Ctrl + F5, o se puede hacer pulsar click en una línea en blanco dentro de la ventana de Watches e introducir la expresión manualmente como se muestra en la siguiente figura.



Una característica nueva de esta ventana en Delphi7 es la capacidad de agrupar las expresiones en pestañas para permitir una depuración más sencilla. Para agregar un nuevo grupo de expresiones se debe pulsar click derecho sobre la ventana de Watches y seleccionar la acción Add Group del menú contextual.

La ventana de Variables Locales

Esta ventana es similar a la ventana de Watches, pero muestra solamente las variables locales y los parámetros recibidos en la rutina evaluada actualmente en el depurador, además no permite agregar expresiones adicionales.

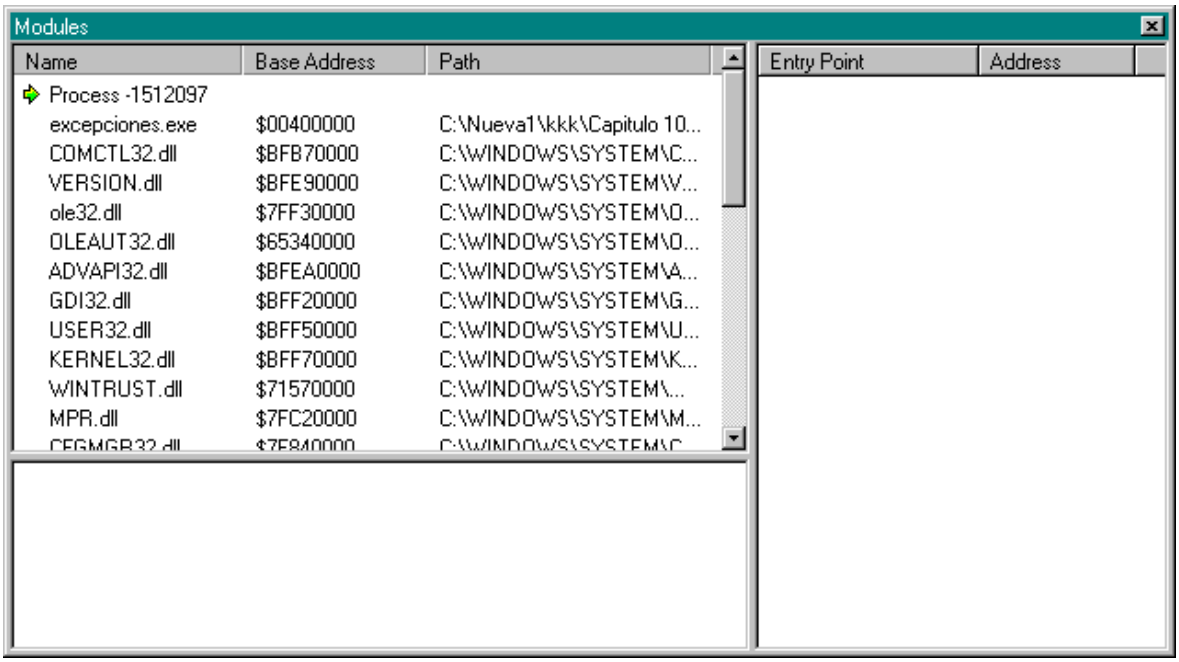
Tanto la ventana de Watches como la de Variables Locales permiten inspeccionar en mayor detalle las clases evaluadas pulsando click derecho sobre la expresión y seleccionando la opción Inspect para desplegar la ventana Debug Inspector, la cual permite obtener, información de los métodos y propiedades definidos para la clase seleccionada.

La ventana de Hilos (Threads)

En una aplicación de múltiples hilos esta ventana permite observar el estatus de los distintos hilos en ejecución.

La ventana de Módulos

Esta ventana permite visualizar que DLLs está utilizando su aplicación como se muestra en la siguiente figura.



Esta ventana contiene tres paneles, en el ángulo superior izquierdo está el panel de módulos, a su derecha está el panel de punto de entrada y finalmente el panel de archivos fuente. El panel de módulos muestra la aplicación actual y las DLLs cargadas, esta lista puede servir para determinar si necesitamos distribuir alguna de estas DLLs con nuestra aplicación, al seleccionar una entrada en esta lista se muestra el punto de entrada en el panel de punto de entrada, además si se dispone del código fuente en Delphi del módulo seleccionado, las unidades de ese módulo se muestran en el panel de archivos fuente.

La ventana de Eventos de Log

La ventana de eventos de log permite visualizar eventos asociados con la aplicación activa, existen cuatro clases de categorías de mensajes que se pueden escribir en el log de eventos:

- **Mensajes de puntos de detención (breakpoints)**

  Se producen cuando se encuentra un punto de detención en el código fuente.

- **Mensajes de procesos**

  Se producen cada vez que un proceso se carga o termina, cada vez que un módulo es cargado o descargado por el proceso.

- **Mensajes de Threads**

  Se producen cada vez que se crea o se destruye un thread durante la sesión de depuración.

- **Mensajes de Modules**

  Se producen cada vez que un módulo (ejecutable, objeto compartido, o paquete) se carga o se descarga. Este incluye el nombre dei módulo, su dirección base, y si posee información de depuración.

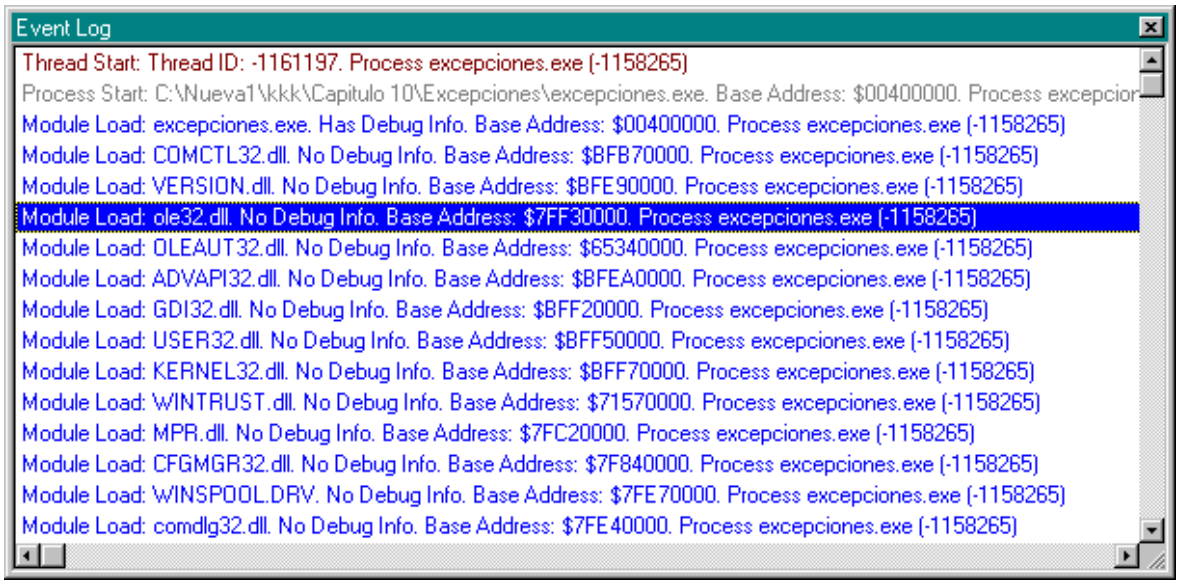
- **Mensajes de salida**

  Se producen cuando se hace una llamada a la API de Windows OutputDebugString.

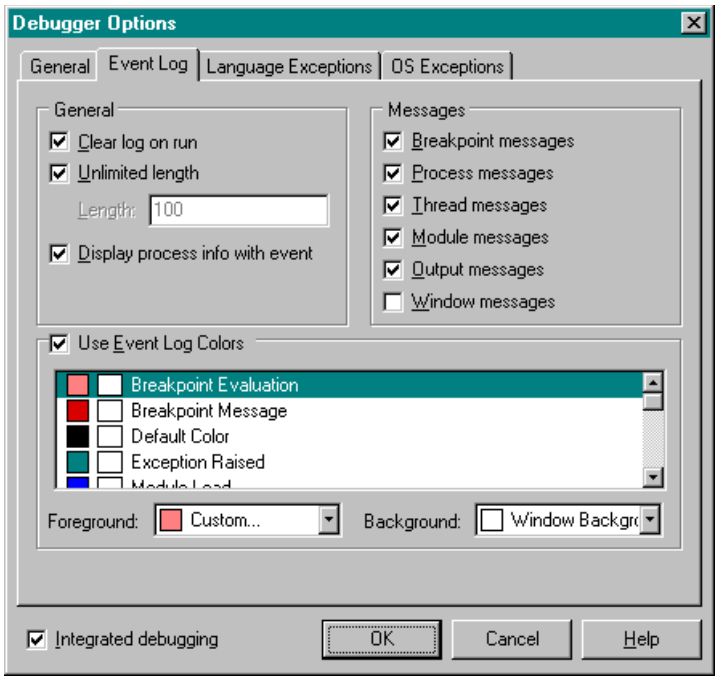
- **Mensajes de Windows**

  Se producen cada vez que un mensaje de Windows es enviado a una de las ventanas de la aplicación.

La ventana de Event Log se muestra en la siguiente figura.

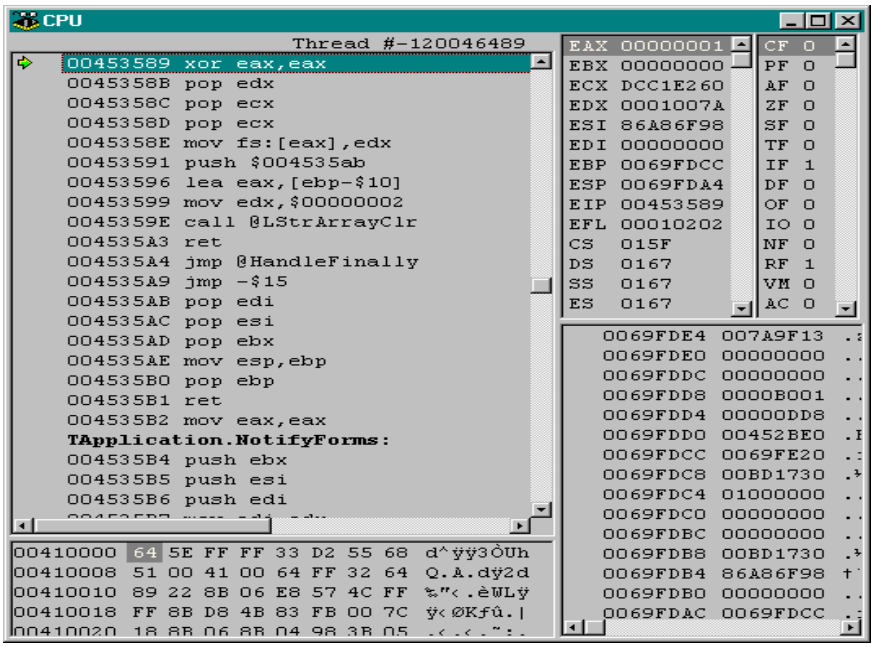


La ventana de Eventos de Log se puede personalizar a través del menú de Delphi Tools | Debugger Options como se muestra a continuación.



La ventana CPU

La ventana de CPU despliega información detallada acerca del código fuente compilado, y los contenidos de los registros de la CPU, esta ventana contiene cinco paneles, en el ángulo superior izquierdo está el panel de desensamblado, a su derecha el panel de registros de CPU, a su derecha el panel de banderas, en el ángulo inferior derecho se encuentra el panel de Stack y a su izquierda el panel de memoria.



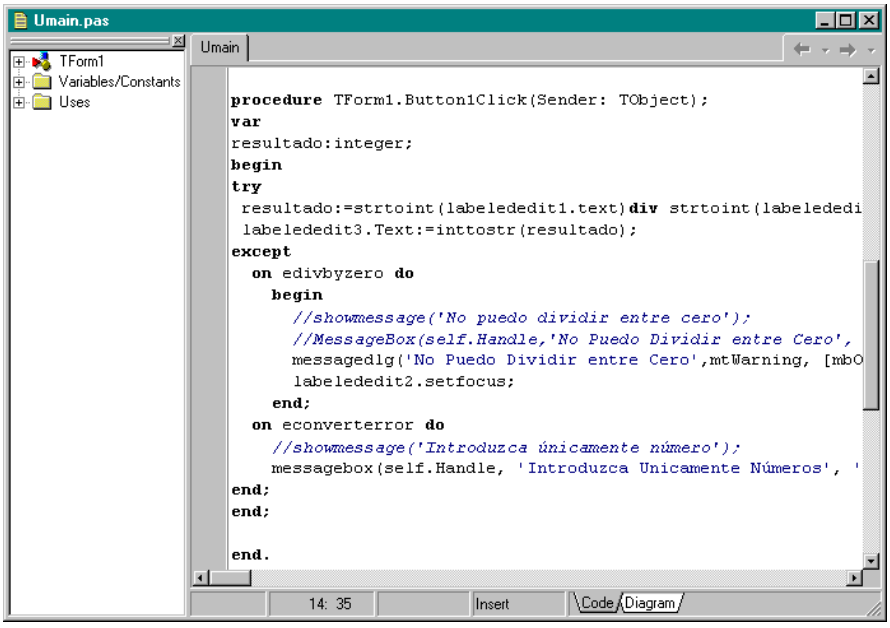
La ventana FPU

Esta ventana muestra los registros de punto flotante del coprocesador matemático (que la mayoría de las computadoras poseen), esta ventana tiene tres paneles, de izquierda a derecha estos son: el panel de los registros FPU, el panel de banderas de control y el panel de las banderas de Status.

10.3 Puntos de Detención

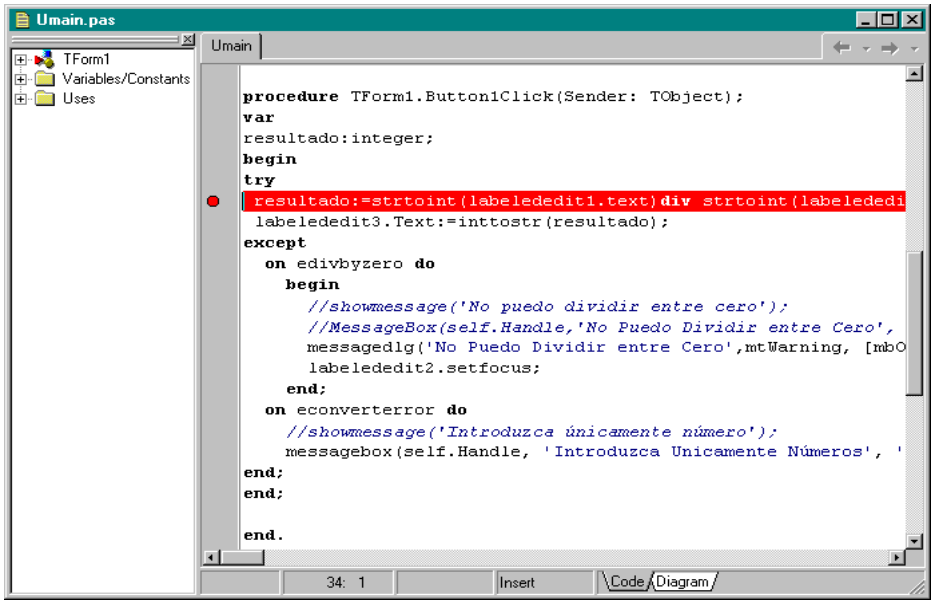
Un punto de detención es un marcador asociado con una línea de código ejecutable o con un evento. Cuando el depurador integrado está habilitado, el depurador evalúa el punto de detención cuando la línea de código que lo contiene está a punto de ser ejecutada u ocurre el evento.

Un punto de detención de código está asociado con una línea ejecutable de código compilado, aunque hay que destacar que por motivos de optimización algunas líneas de código fuente no compilan y por lo tanto no son ejecutables, por lo tanto si se asocia un punto de detención a una línea no ejecutable este nunca se disparará.
Después de compilar un proyecto de Delphi, las líneas que son compiladas pueden identificarse por un pequeño diamante azul que aparece en el borde izquierdo del editor de código, tal como se muestra en la siguiente figura.

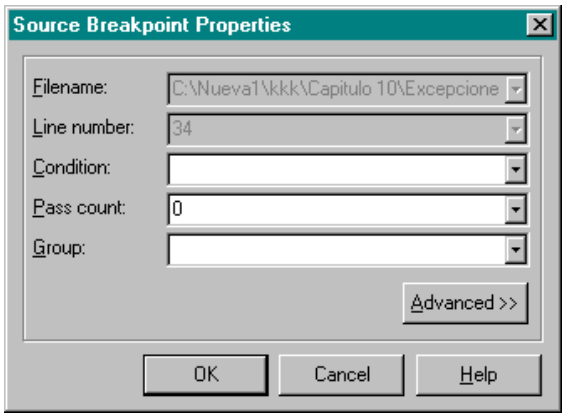


Para agregar un punto de detención se puede pulsar click en el borde izquierdo del editor de código a la altura de la línea de código donde se desea colocar el punto de detención, o bien estando posicionado en el editor de código en la línea deseada se puede presionar la tecla F5.

Los puntos de detención se muestran en el editor de código con un punto rojo en lugar del diamante azul en el borde derecho del editor, además la línea asociada está seleccionada con un color rojo (configuración por defecto).



Una vez que se ha establecido un punto de detención se puede controlar su comportamiento a través del cuadro de dialogo Source Breakpont Properties, el cual se muestra al pulsar click con el botón derecho del mouse sobre el circulo rojo del punto de suspensión y seleccionando la opción Breakpoint properties del menú contextual que se presenta.



Por defecto un nuevo punto de detención es incondicional, es decir cuando el depurador ejecuta la línea de código que contenga el punto de detención, la ejecución se detiene justo antes de ejecutar la línea de código. Sin embargo los puntos de detención de código pueden personalizarse a través del cuadro de diálogo anteriormente mostrado, logrando que el depurador se detenga en base a una condición booleana. o después de ejecutar la línea de código cierto número de veces, o inclusive se puede declarar que el punto de detención pertenece a un grupo de puntos de detención. A continuación describiremos en mayor detalle las secciones de este cuadro de diálogo.

Nombre de Archivo y Numero de Línea

Los cuadros de edición Filename y Line number despliegan el nombre del archivo y el número de línea de código al cual está asociado el punto de suspensión, estos valores están fijos y no necesitan modificarse.

Condiciones

Se puede utilizar el cuadro de texto Condition para introducir una expresión booleana de tal manera que el depurador solamente se detenga en el punto de suspensión cuando la expresión se evalúe como verdadera.

La única limitación de esta expresión condicional es que todos los símbolos utilizados en dicha expresión deben de estar dentro del alcance del punto de suspensión. Por ejemplo, se puede utilizar una variable local en la condición la cual esté definida en el método que contiene al punto de suspensión. Similarmente se puede utilizar

una función en la expresión siempre y cuando el punto de suspensión se encuentre dentro de una unidad que utilice dicha función.

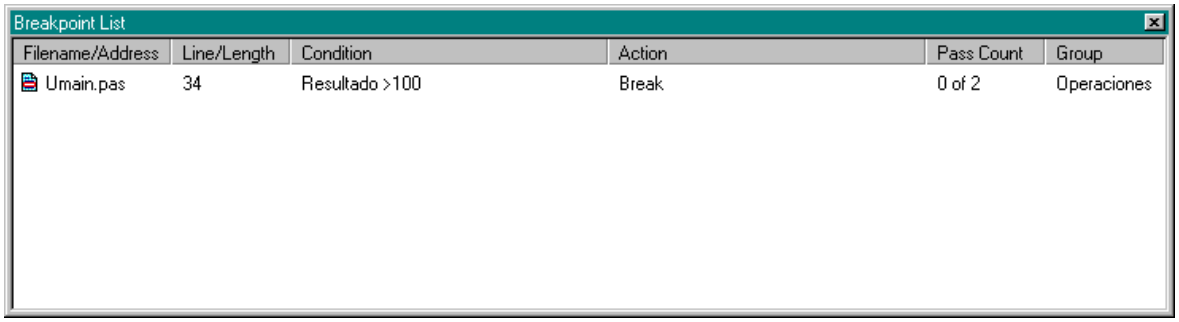
Conteo de pasadas

El cuadro de texto Pass count del cuadro de diálogo de las propiedades de un punto de suspensión permite indicarle al depurador que detenga la ejecución del programa en el punto de suspensión hasta que este se haya ejecutado cierto número de veces. Por ejemplo si tuviéramos un ciclo de 100 iteraciones que sabemos que se ejecuta correctamente 50 veces y falla después, podemos indicarle al depurador que se detenga después de 50 ejecuciones de la línea de código asociada al punto de suspensión.

Grupos de puntos de suspensión

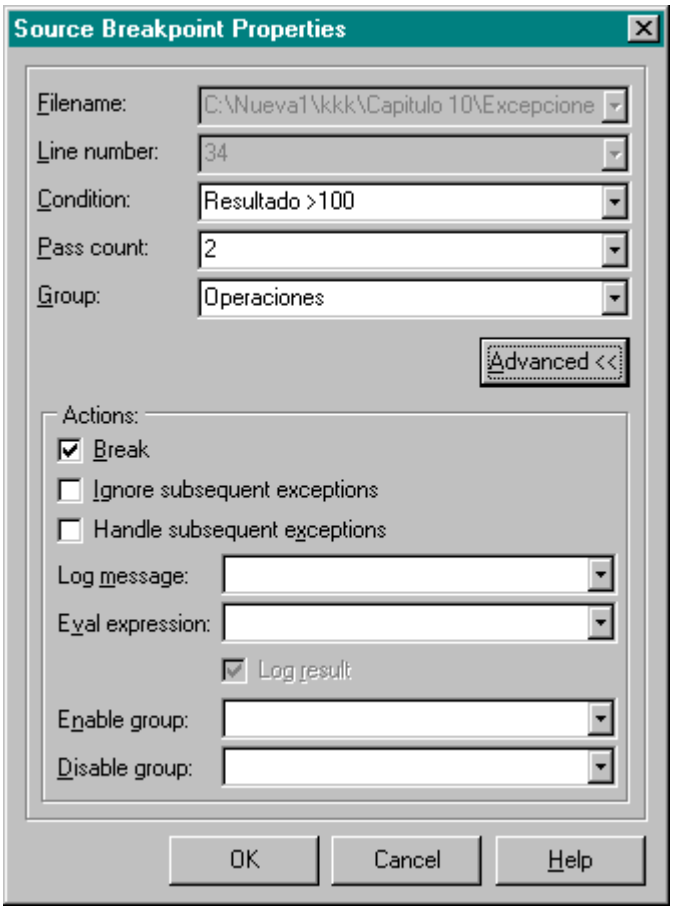
Los puntos de suspensión se pueden organizar en grupos, esto con el objetivo de deshabilitar un grupo de puntos de suspensión en tiempo de ejecución. Para asignar un punto de suspensión a un grupo, especifique el nombre del grupo en la lista de selección del cuadro de diálogo de las propiedades del punto de suspensión.

La ventana de puntos de suspensión muestra las propiedades definidas para los puntos de suspensión del proyecto actual tal como se muestra en la siguiente figura.



Acciones de puntos de suspensión

Los puntos de suspensión también se pueden personalizar a través de acciones, para desplegar las acciones de un punto de suspensión pulse click en el botón Advanced del cuadro de diálogo de propiedades del punto de suspensión, esto expande el cuadro de diálogo, tal como se muestra en la siguiente figura



La casilla de verificación Break permite especificar si el punto de suspensión invocará al depurador integrado o no, cuando está verificada (la opción por defecto), el depurador se carga cuando el punto de suspensión se ejecuta (dependiendo de otras propiedades como condiciones y conteo de pasadas). Cuando la casilla no está verificada, el depurador no se cargará y no se detendrá la ejecución del programa.

Sin embargo aunque la ejecución del programa no se detiene, el punto de ejecución todavía se ejecuta, así que cualquier otra acción definida por el punto de suspensión se ejecutará, lo cual hace de los puntos de suspensión de Delphi una poderosa herramienta de depuración.

Una de las acciones que pueden llevar a cabo los puntos de suspensión que no detienen la ejecución es habilitar o deshabilitar las excepciones subsecuentes, como mencionamos anteriormente, el depurador de Delphi puede ignorar algunas o todas las excepciones generadas, sin embargo esto deshabilitaria las excepciones para todo el proyecto actual; a través del uso de puntos de suspensión se pueden deshabilitar las excepciones para puntos

específicos de una aplicación, esto se logra utilizando un par de puntos de suspensión uno indicándole al depurador que ignore las excepciones y otro habilitándolas.

Por ejemplo suponga que tenemos una rutina que lanza explícitamente excepciones con el propósito de detectar condiciones de error, si no desea que esta rutina detenga la ejecución de la aplicación cuando se está depurando, podemos colocar dos puntos de suspensión, ambos que no detengan la ejecución del programa, el primero se colocaría al inicio de la rutina indicando que ignoren las excepciones subsecuentes, y el segundo se colocaría al final de la rutina, indicando que no ignore las excepciones.

Otra acción que pueden ejecutar los puntos de suspensión es escribir mensajes al log de mensajes descrito anteriormente, para escribir un mensaje estático en el log, escríbalo en la lista de selección Log message del cuadro de diálogo de propiedades avanzadas del punto de suspensión, también es posible escribir el valor de una expresión escribiéndola en la lista de selección Eval expresión, la casilla de verificación Log result determina si el valor de la expresión se escribe en el log o no, en ocasiones puede ser deseable ejecutar una expresión que proporcione efectos colaterales sin escribir el valor de la expresión al log.

Un punto de suspensión puede habilitar o deshabilitar un grupo completo de puntos de suspensión, esto se logra a través de las listas de selección Enable group, la cual permite indicar el grupo a habilitar y Disable group, la cual permite indicar el grupo a deshabilitar.

10.4 Preguntas y Ejercicio de Repaso

1. ¿Cuál es la función de la ventana de depuración Call Stack?

2. Menciona los tres tipos básicos de Errores

3. Cual de estas opciones no pertenece a la ventana de Depuración.

a) Breakpoints b)Run c)Watches d)Modules e)CPU

4. Describa las características del manejo de Excepciones en Delphi.

5. Describa la función de los puntos de suspensión en Delphi.

6. Mencione las opciones de configuración de los puntos de suspensión de código.

7. Para que me es util dentro de la programación un Breakpoints.

8. Desde tu punto de vista cual ventana de depuración es mas completa para ver el funcionamiento del sistema.

9. Realiza la suma de varias variables dentro de un ciclo y utilizando una ventana de depuración observa como se van comportando tus variables.

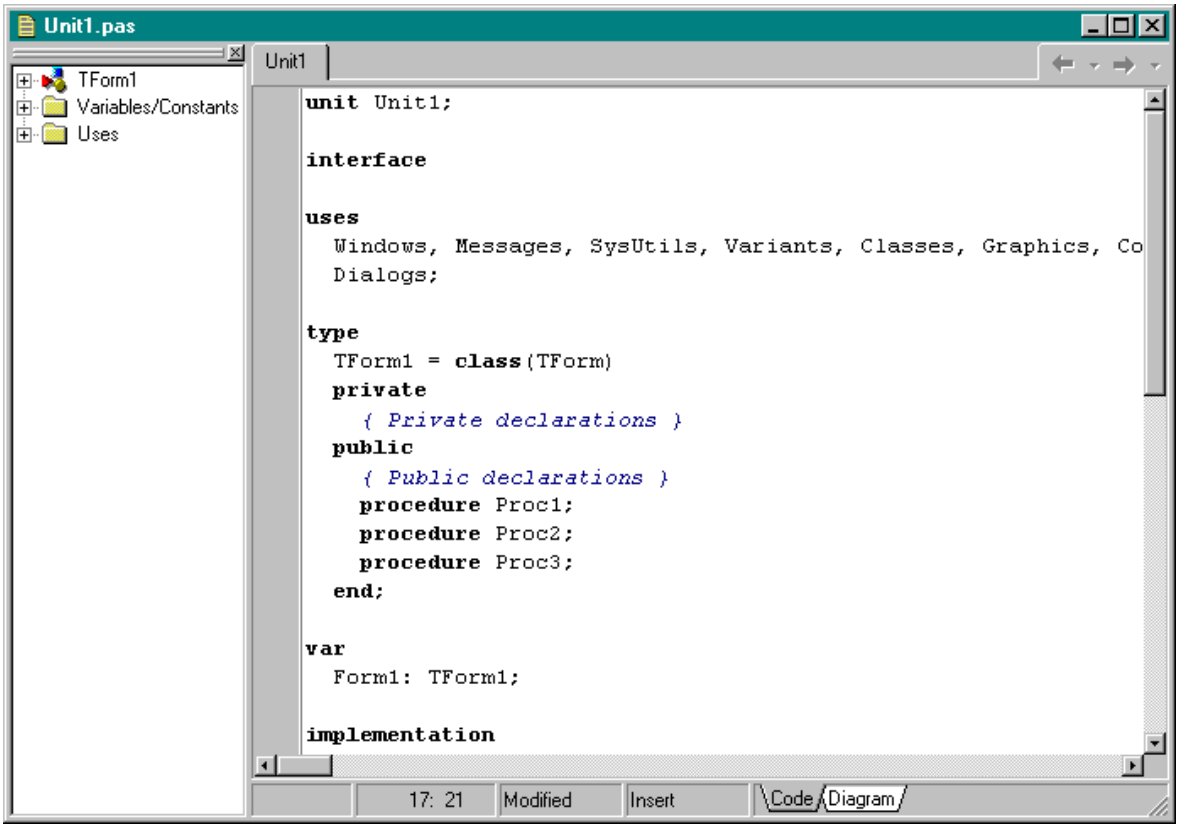
Ejercicio de Repaso

1. Genere una nueva aplicación mediante el menú File | New | Application.
2. Salve su proyecto con los siguientes nombre de archivo:

Archivo de la forma: UMain.pas

Archivo del proyecto: Excepciones.dpr

3. Agregue un componente Button (pestaña Standard) a la forma, modifique su propiedad Caption al valor &Calcula.
4. Dentro de la unidad Umain, agregue tres procedimientos a la sección public de la clase de la forma; la declaración de su forma deberá verse como la siguiente figura.



5. Pulse la combinación de teclas Ctrl + Shift + C para generar el cuerpo de los procedimientos en la sección implementación. Asigne a cada uno de ellos el código que se indica a continuación.

```
procedure TForm1.Proc1;
begin
  try
    proc2;
```

```
  except
    on EZeroDivide do
      ShowMessage('No es posible dividir entre cero!');
  end;
end;
procedure TForm1.Proc2;
begin
  proc3;
end;
procedure TForm1.Proc3;
var
x, y, z: Double;
begin
  x:= 10;
  y:= 0;
  z:= x/y;
end;
```

Agregue el siguiente código en el manejador de! evento OnClick del componente Button1.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 proc1;
end;
```

El código del evento OnClick del botón ejecuta el procedimiento Proc1, el cual a su vez ejecuta el procedimiento Proc2 dentro de un bloque try..except. El procedimiento Proc2 ejecuta el procedimiento Proc3, el cual genera una excepción del tipo EZeroDivide, entonces Delphi busca recursivamente en la pila de llamadas un bloque try..except, el cual se encuentra en Proc1, el control se transfiere al manejador de la excepción y finalmente la excepción es manejada.

Al ejecutar la aplicación, Delphi mostrará el siguiente cuadro de diálogo, siempre y cuando se encuentre habilitada la opción Stop on Delphi Exceptions del menú de Delphi Tools | Debugger Options| Lenguage Exceptions.

Si se desmarca la casilla de verificación stop on delphi exceptions, solamente deberán mostrarse los mensajes de error que se definieron en el bloque try..Except..end de la aplicación.

Al manejar la excepción se mostrara el mensaje que nosotros definimos, notificado que ocurrió un error.

XI. Creación de Reportes con Rave Reports.

11.1 Introducción

"Rave Reports es un diseñador visual que ofrece múltiples características que le permiten generar reportes de una manera rápida y eficiente apoyándose en un ambiente de diseño visual que cuenta con una amplia variedad de formatos de informe y múltiples tecnologías que le permiten generar cambios y facilitan el mantenimiento de sus reportes.

Delphi 7 permite emplear Rave Reports en aplicaciones VCL y CLX para generar reportes desde una base de datos y con información independiente.

La página Rave, de la paleta de componentes de Delphi, provee los componentes que permiten generar reportes en sus aplicaciones.

Existen dos tipos distintos de objetos en Rave Reports: Componentes de Salida y Clases de Reportes.
Los Componentes de salida son responsables de enviar el reporte a una variedad de destinos, mientras que las Clases de Reportes manejan otras tareas del reporte.

11.2 Componentes de Salida

TRvSystem. Incorpora una impresora estándar y un sistema de previsualización en un componente fácil de utilizar.

TRvNDRWriter. Genera un stream NDR a un archivo (en un formato propietario) desde la ejecución del reporte.

TRvRenderPreview. Despliega un diálogo de vista previa para un stream NDR o archivo.

TRvRenderPrinter. Envía un stream NDR o archivo a la impresora.

TRvRenderFDF. Convierte un stream NDR o archivo a formato PDF.

TRvRenderHTML Convierte un stream NDR o archivo a formato HTML.

TRvRenderRFT. Convierte un stream NDR o archivo a formato RFT.

TRvRenderText. Convierte un stream NDR o archivo a formato Texto.

11.3 Clases Rave

TRvProject. Proporciona una conexión a un proyecto de reporte que fue creado con el diseñador visual Rave. Este componente se utiliza para obtener una lista de todos los reportes disponibles o para ejecutar un reporte en particular.

TRvCustomConnection. Conecta datos personalizados (generados a través de eventos) DirectDataViews creados con el diseñador visual Rave.

TRvDataSetConnection. Conecta un componente TDataSet a un DirectDataViews creado con el diseñador visual Rave.

TRvTableConnection. Conecta componentes TTable a DirectDataViews creados con el diseñador visual Rave.

TRvQueryConnection. Conecta componentes TQuery a Direct Data Views creados con el diseñador visual Rave."[1]

[1] http://www.nevrona.com

Aplicación de Ejemplo

1. Cree una nueva aplicación con el menú de Delphi File | New | Application.
2. Cree un data module por medio del menú de Delphi File | New | Data Module, salve sus archivos con los siguientes nombres:

Archivo del data module: UDataModule.pas

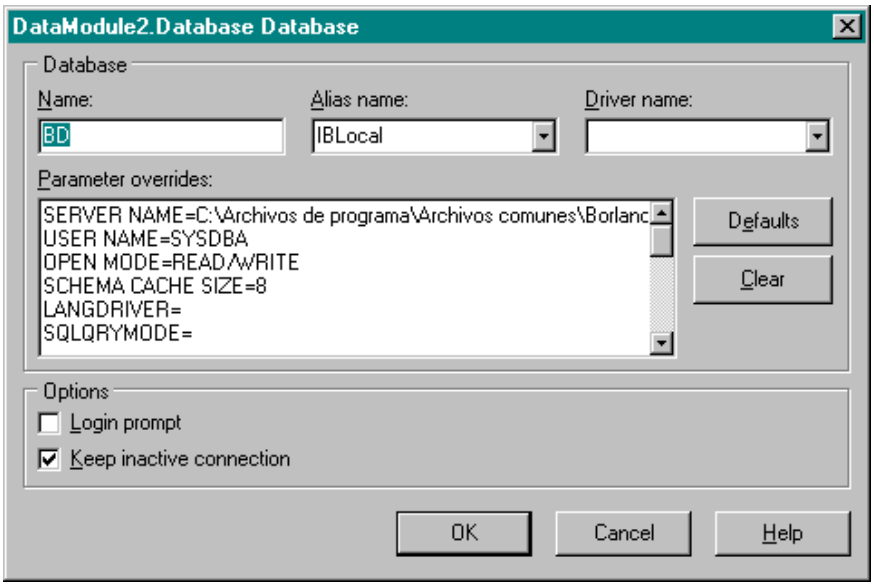
Archivo de la forma: UMain.pas

Archivo del proyecto: DemoRave.dpr

3. Coloque en el data module un componente TDatabase (pestaña BDE) y modifique sus propiedades como se indica a continuación.

PROPIEDAD	VALOR
Name	Database
DatabaseName	DB

4. Pulse doble click sobre el componente Database, dentro del cuadro de diálogo de configuración seleccione en la opción AliasName el alias IBLocal, pulse click en el botón Defaults para mostrar los parámetros del alias y especifique que el valor del password es masterkey, finalmente desmarque el checkbox Login prompt El cuadro de configuración deberá verse tal como se muestra en la siguiente figura.



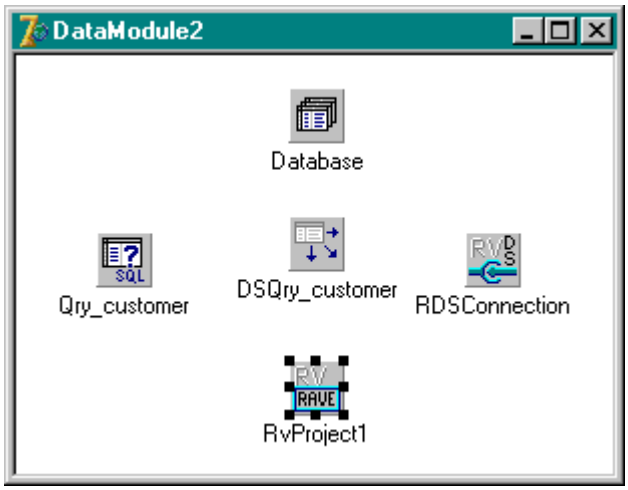
5. Modifique la propiedad Connected del componente Database al valor True.
6. Coloque en el data module un componente TQuery (pestaña BDE) y modifique sus propiedades como se indica a continuación.

PROPIEDAD	VALOR
Name	Qry_Customer
DatabaseName	DB
SQL	SELECT * FROM CUSTOMER
Active	True

7. Coloque en el data module un componente TDataSource (pestaña Data Access) y modifique su propiedad DataSet al valor Qry_Customer y su propiedad Name a DSQry_Customer.
8. Coloque en el data module un componente TRvDataSetConnection (pestaña Rave) y modifique las siguientes propiedades.

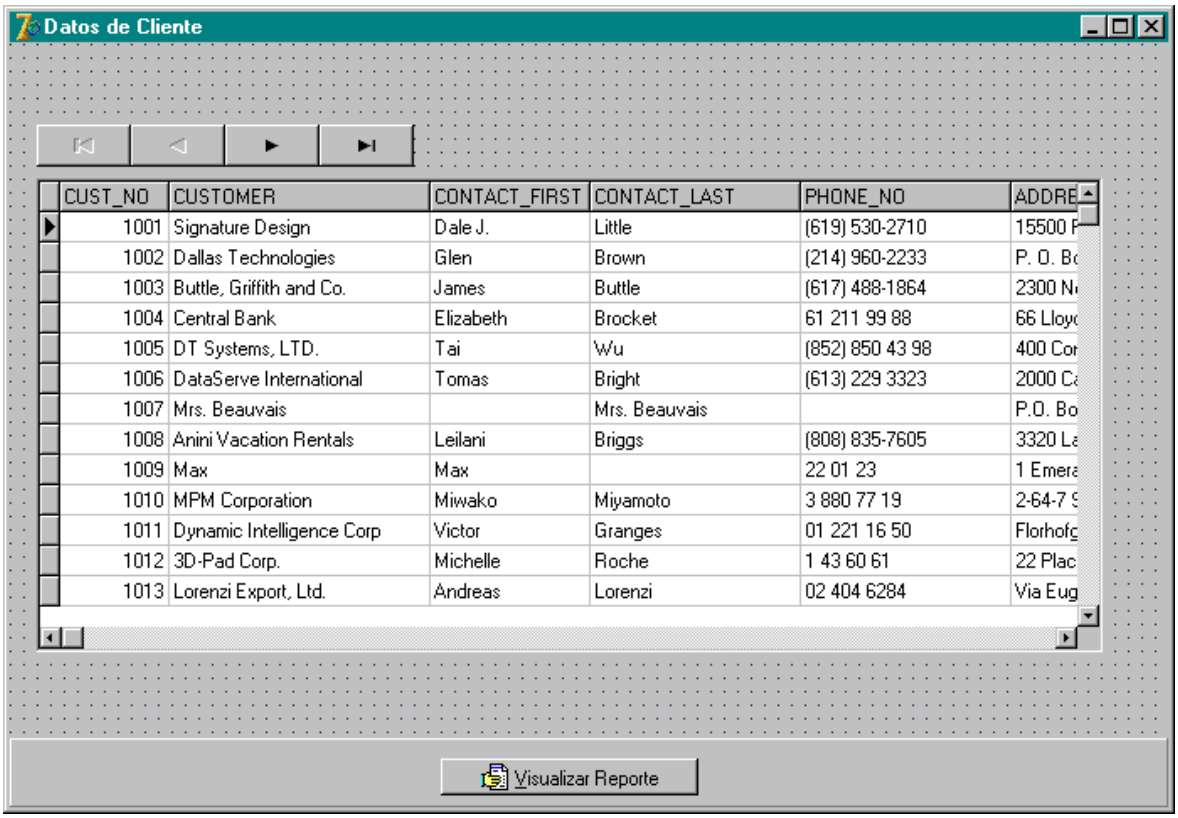
PROPIEDAD	VALOR
Name	RDSConnection
DataSet	Qry_Customer

9. Coloque en el data module un componente TRvProject (pestaña Rave).



10. Visualice la forma Form1 y del menú de Delphi seleccione la opción File | Use Unit y seleccione la unidad UDatamodule dentro del cuadro de diálogo Use Unit. Pulse click en el botón OK.

A continuación configuraremos la forma principal del proyecto, como se muestra en la siguiente figura:

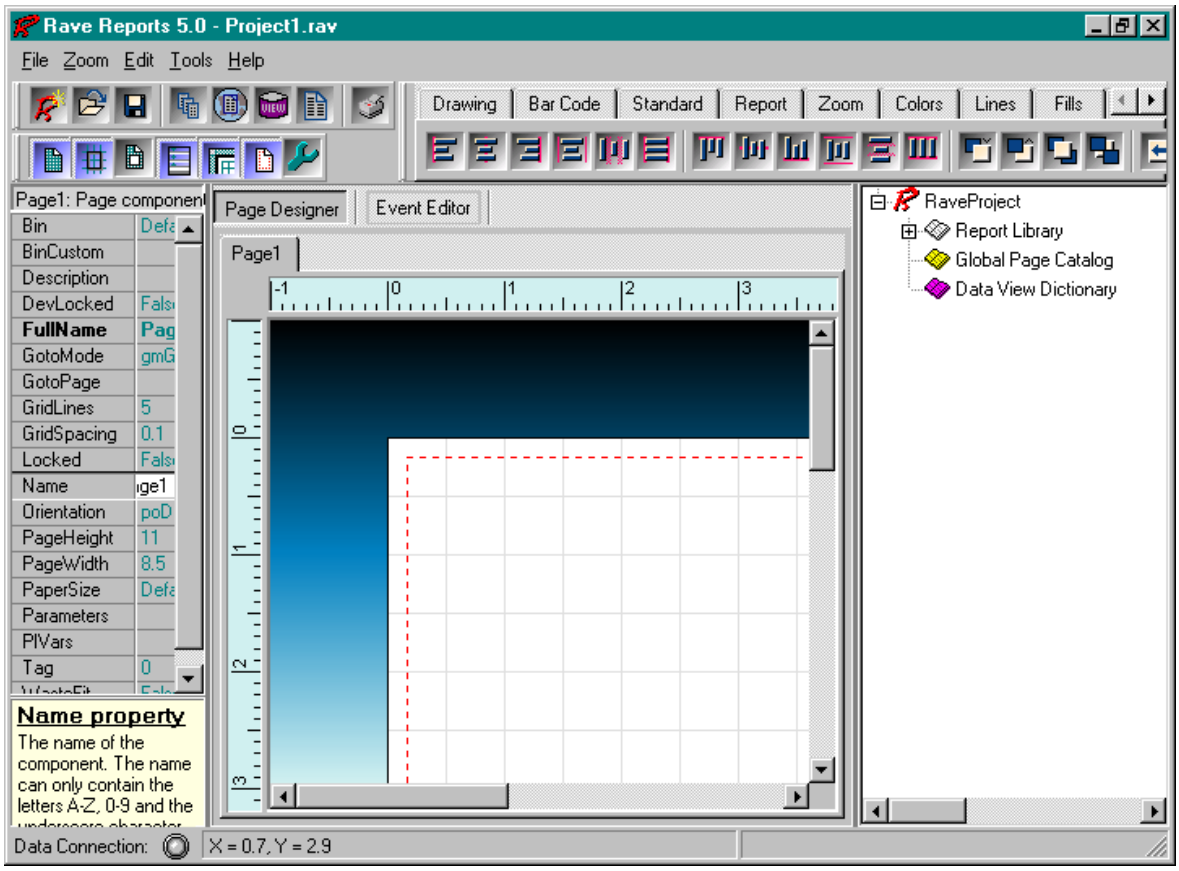


11. Modifique la propiedad Name de la forma al valor FrmMain y su propiedad Caption al valor Datos de Clientes.
12. Agregue la forma un componente TDBNavigator y un componente TDBGrid (pestaña Data Controts) y modifique la propiedad DataSource de ambos componentes al valor DataModule2.DSQry_Customer.
13. Coloque un componente TPanel (pestaña Standard) a la forma, modifique su propiedad Align al valor alBottom y elimine el contenido de su propiedad Caption.

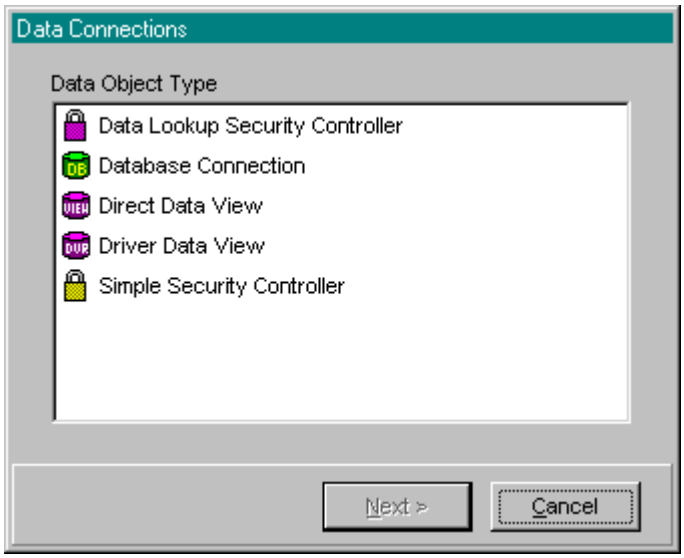
14. Agregue dentro del componente Panel1 un componente TspeedButton modifique su propiedad Caption al valor &Visualizar Reporte y su propiedad Glyph asóciela al archivo Report.bmp que se encuentra en la ruta:

C: \ Archivos de programa\ Archivos cornunes\Borland Shared\Images\Buttons

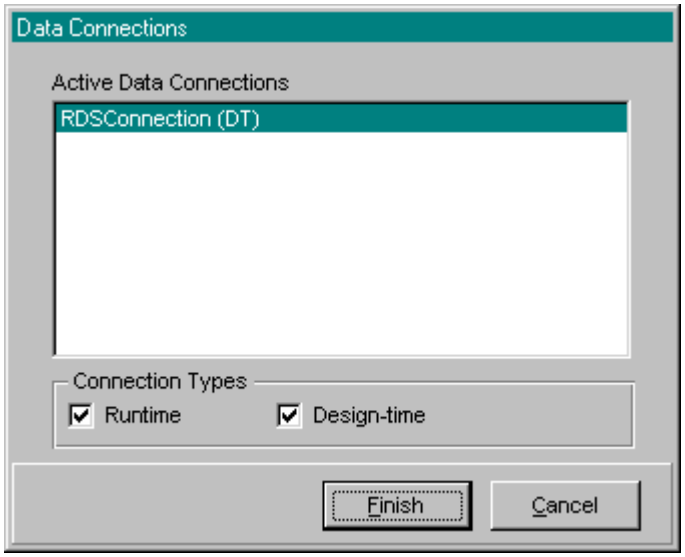
15. Regrese al data module y pulse doble click sobre el componente RvProject1 para visualizar el deseñador visual del reporte que se muestra en la siguiente figura.



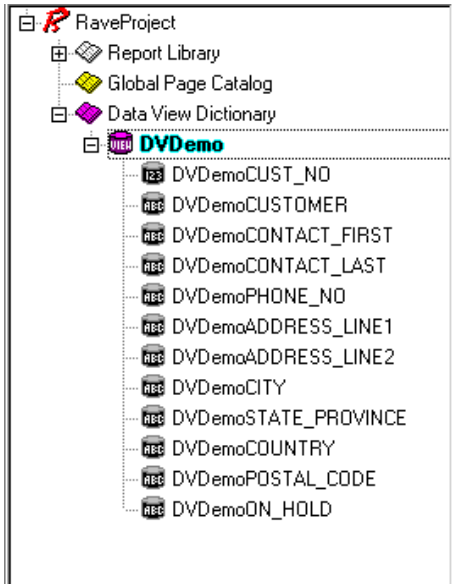
16. En el diseñador visual, seleccione la opción File | New Data Object para mostrar el cuadro de diálogo Data Connections, como se muestra en la siguiente figura.



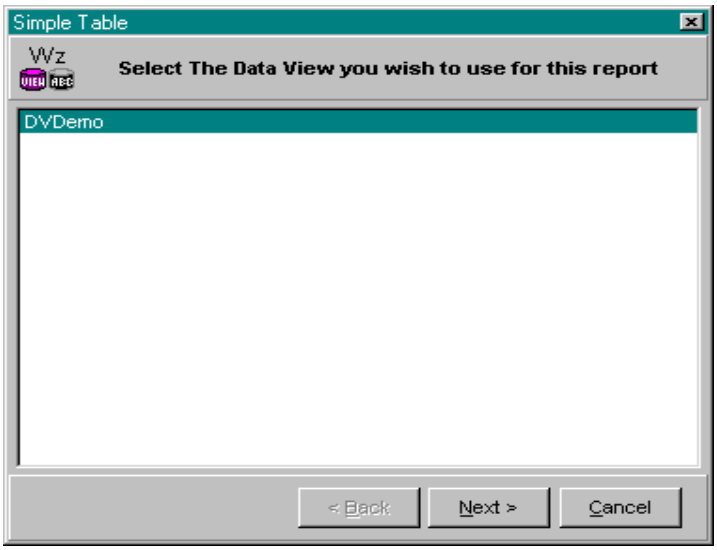
17. Dentro de este cuadro de diálogo, seleccione la opción Direct Data View y pulse click en botón Next,
18. En el cuadro de diálogo que se presenta a continuación asegúrese de que la opción RDSConnection este seleccionada y pulse click en el botón Finish.



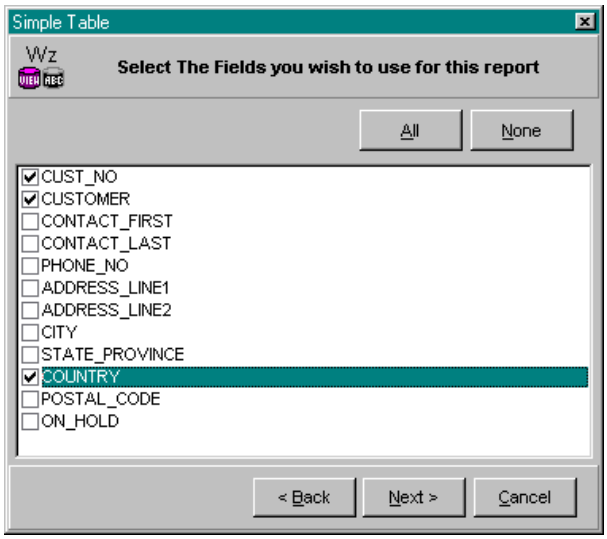
19. Dentro del diseñador visual de Reportes, localice el árbol de proyectos, que se ubica en el extremo derecho de su pantalla y expanda el nodo Data View Dictionary, seleccione el nodo DataViewl, y utilizando el panel de propiedades, ubicado en el extremo izquierdo de la pantalla, modifique la propiedad Name al valor DVDemo.



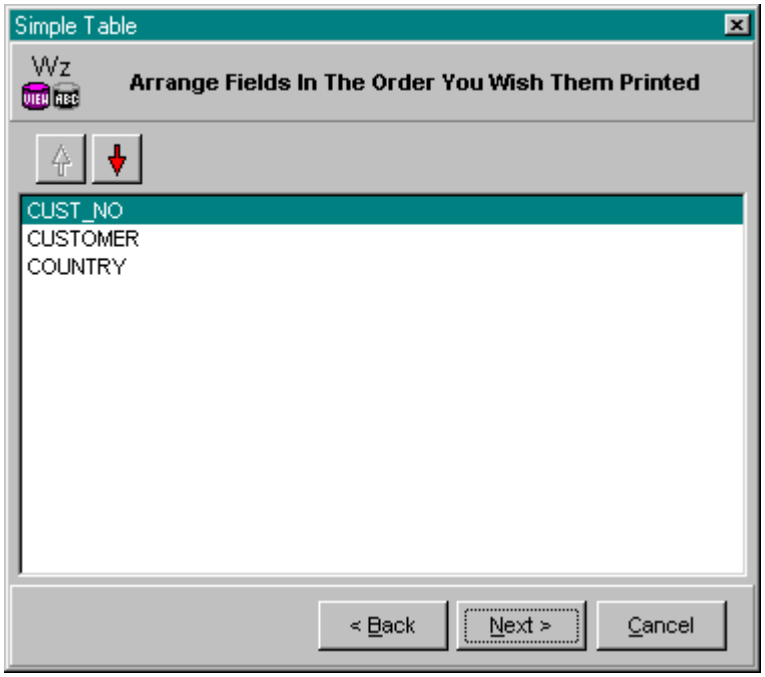
20. Ahora estamos listos para crear nuestro reporte, seleccione el menú Tools | Report Wizards | Simple Table para invocar el asistente mostrado en la siguiente figura.



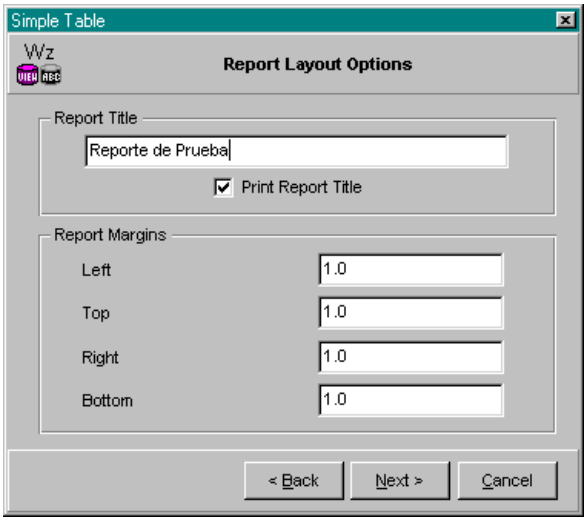
21. Asegúrese de que la opción DVDemo esté seleccionada y pulse click en el botón Next. En el cuadro de diálogo que se presenta a continuación, seleccione dos o tres campos que se visualizarán en el reporte y presione click en el botón Next, tal como se muestra a continuación.



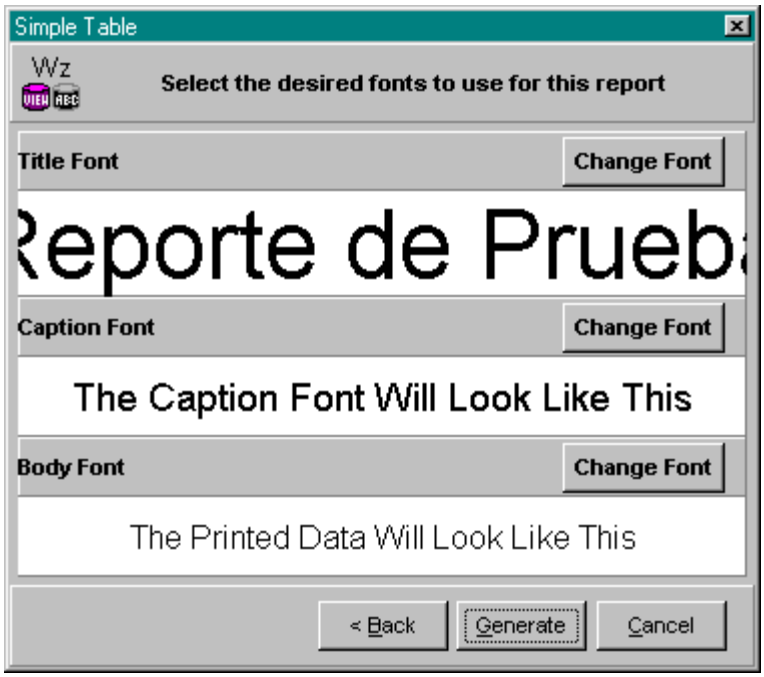
22. En el siguiente cuadro, cambie el orden de los campos si lo desea y presione click en el botón Next



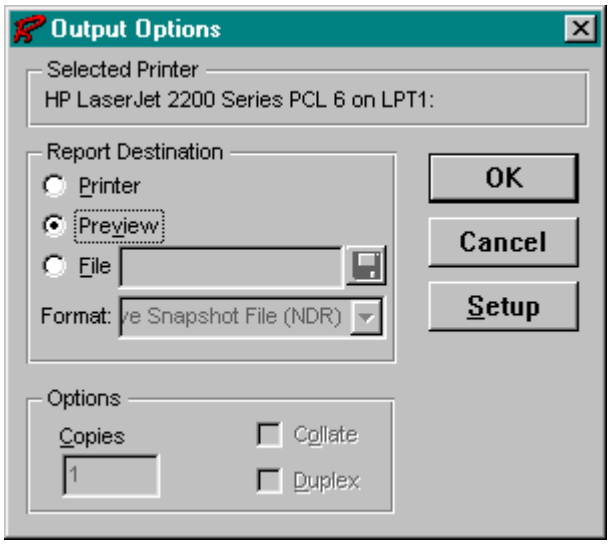
23. A continuación, modifique él titulo del reporte con el dato Reporte de Prueba, como se muestra en la siguiente figura y presione click en el botón Next.



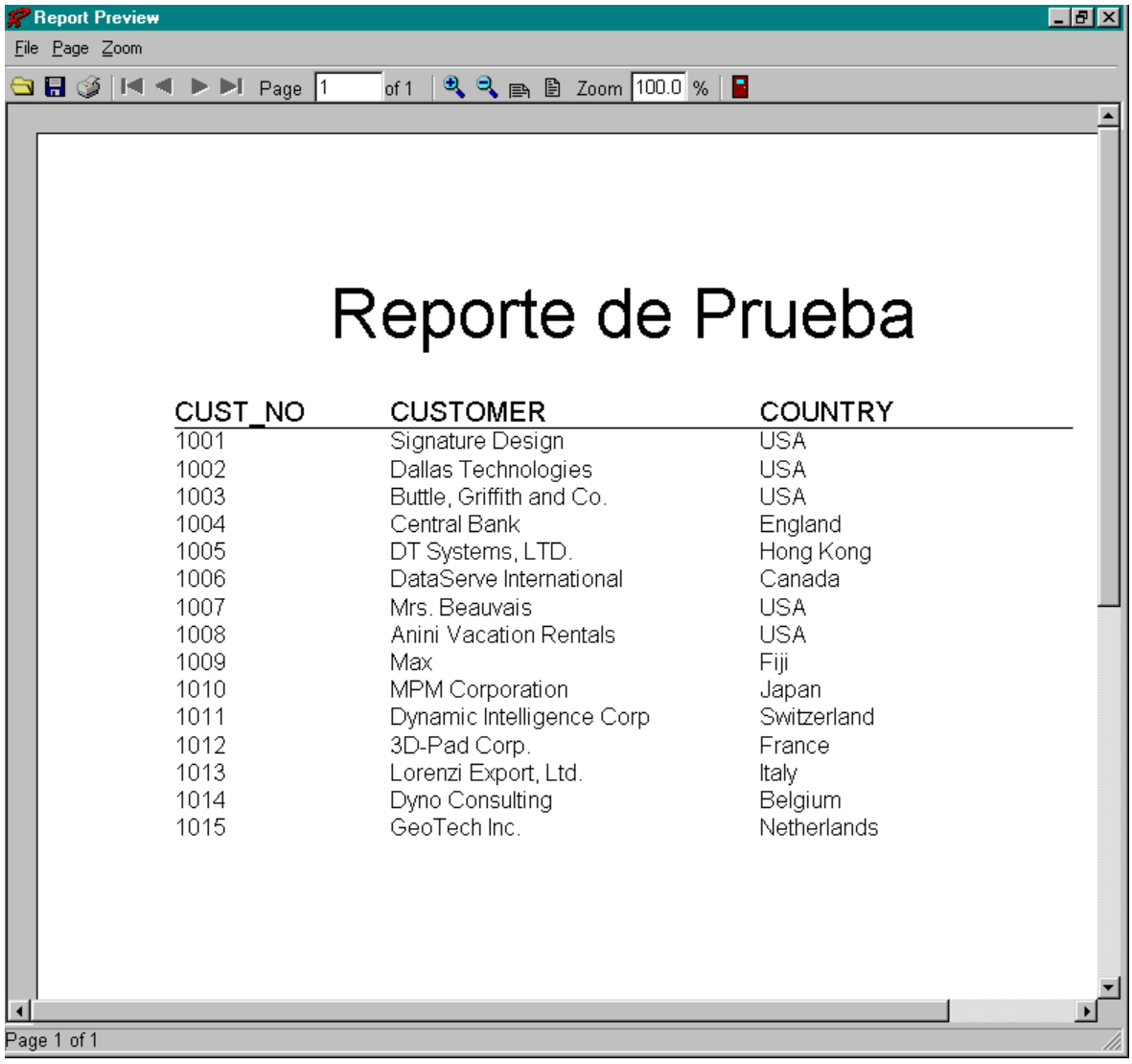
24. Por último, modifique el formato de las fuentes de los títulos de su reporte, si lo desea y después presione click en el botón Generate.



25. Para mostrar la vista previa del reporte, seleccione el menú File | Execute Report para mostrar el cuadro de diálogo de la siguiente figura.



26. Dentro del cuadro Output Options, asegúrese de que la opción Preview este seleccionada como el destino del reporte y presione click el botón OK para mostrar la vista previa del reporte.



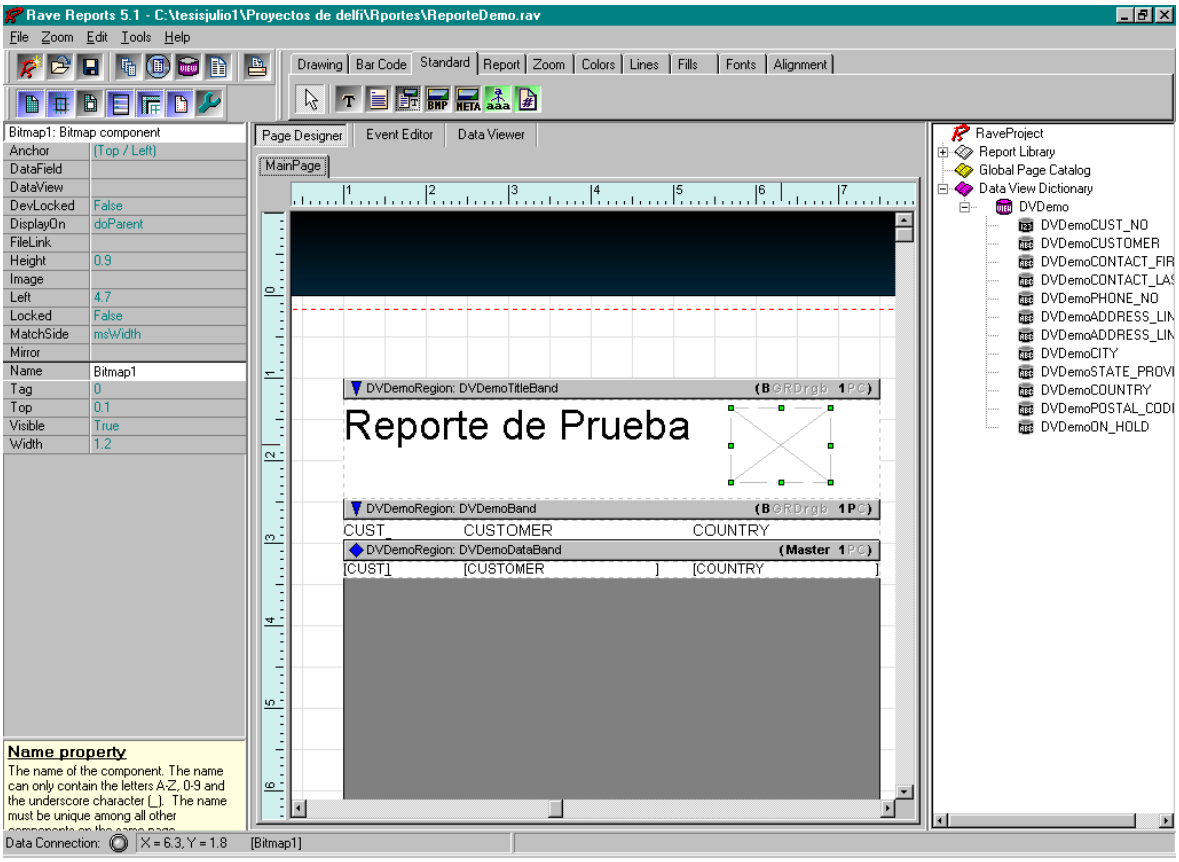
27. Cierre la pantalla de la vista previa del reporte.
28. Grabe el proyecto de Reporte como ReporteDemo.rav.

Asignando Formato al Reporte

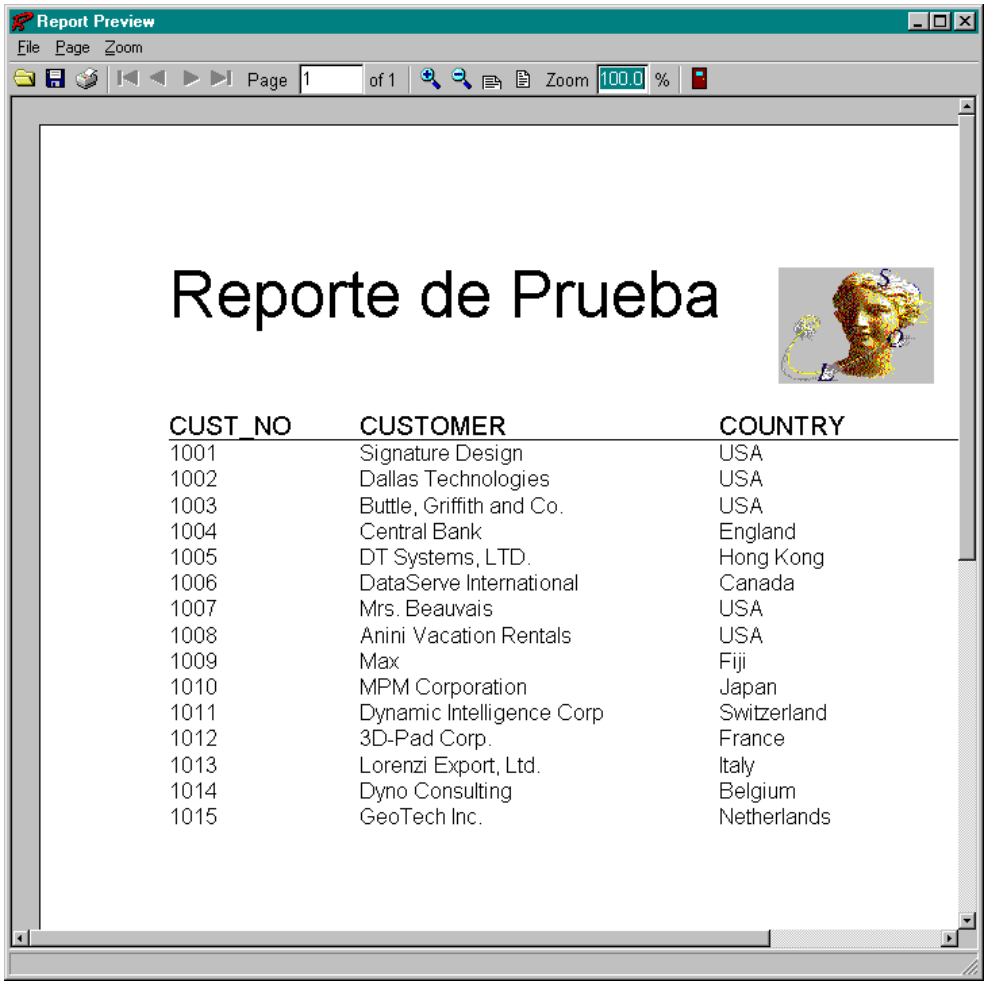
29. Dentro del área del reporte seleccione el componente Text Reporte de Prueba que se ubica en la banda DVDemoDataBand y modifique su propiedad FontJustify por el valor pjLeft.
30. De la pestaña Standard, del diseñador visual de reportes, seleccione el componente Bitmap y colóquelo en el extremo derecho de la banda DVDemoDataBand.

31. Modifique la propiedad Image del componente Bitmap1 por:

C:\Archivos de programa\Archivos comunes\Borland Shared\Irnages\Splash\l6Color\athena.bmp



32. Modifique la propiedad MatchSide del componente Bitmap1 al valor msBoth, para ajustar la imagen al tamaño del componente.
33. Grabe los cambios del reporte y pulse la tecla de función F9 para visualizarlo.



CUST_NO	CUSTOMER	COUNTRY
1001	Signature Design	USA
1002	Dallas Technologies	USA
1003	Buttle, Griffith and Co.	USA
1004	Central Bank	England
1005	DT Systems, LTD.	Hong Kong
1006	DataServe International	Canada
1007	Mrs. Beauvais	USA
1008	Anini Vacation Rentals	USA
1009	Max	Fiji
1010	MPM Corporation	Japan
1011	Dynamic Intelligence Corp	Switzerland
1012	3D-Pad Corp.	France
1013	Lorenzi Export, Ltd.	Italy
1014	Dyno Consulting	Belgium
1015	GeoTech Inc.	Netherlands

34. Cierre la pantalla de la vista previa del reporte y cierre la ventana de Rave Reports para regresar al entorno de desarrollo de Delphi.
35. Dentro del data module localice el componente RvProject1 y modifique su propiedad ProjectFile por ReporteDemo.rav, que corresponde al archivo del proyecto de reporte que realizamos anteriormente.
36. Visualice la forma FrmMain, seleccione el componente SpeedButton1 y por medio del Inspector de Objetos agregue el siguiente manejador para el evento OnClick.

```
procedure TFrmMain.SpeedButton1Click (Sender; TObject);
begin
```

DataModule2.RvProject1.Execute ;

end;

37. Grabe los cambios de la aplicación y ejecútela para probar su funcionalidad.

Construyendo un Reporte Maestro Detalle

Ahora que hemos creado nuestro primer reporte Rave, crearemos un segundo ejemplo integrando nuestro reporte con una aplicación.

Comenzaremos con un reporté basado en la base de datos DB. Este reporte mostrará una relación Maestro Detalle entre las tablas clientes y abonos.

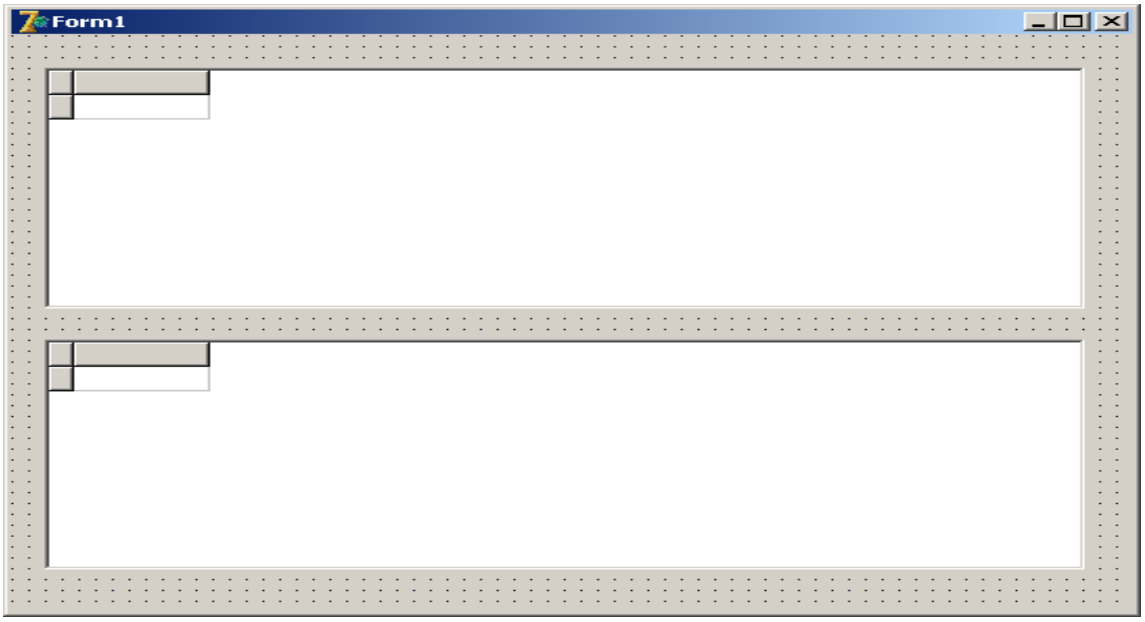
1. Cree una nueva aplicación el menú de Delphi File | New | Application, agregue un data module, File | New | Data Module, salve sus archivos con los siguientes nombres.

Archivo del data module: DataModule.pas

Archivo de la forma; Reporte.pas

Archivo del proyecto: Proyecto.dpr

2. Agregue a la forma dos componentes dbgrid (pestaña Data Controls)



3. Agregamos dos speedbuttons (pestaña Additional), a uno en el caption le ponemos Reporte General y al otro Salir.

4. En el Datamodule agregamos un Database, 2 Ttable(pestaña BDE) y 2 DataSource, modifique sus propiedades como se muestra a continuación.

COMPONENTE	PROPIEDAD	VALOR
Tdatabase	Name	DB
	AliasName	Smue
	DatabaseName	Database
	Connected	True
Table1	Name	Tclientes
	DatabaseName	DB
	TableName	Clientes.db
Table2	Name	Tabonos
	DatabaseName	DB
	TableName	cxccli.db

5. Pulse doble click sobre el componente Salir y por medio del Inspector de Objetos cree el siguiente manejador para el evento OnClick.

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
close;
end;
```

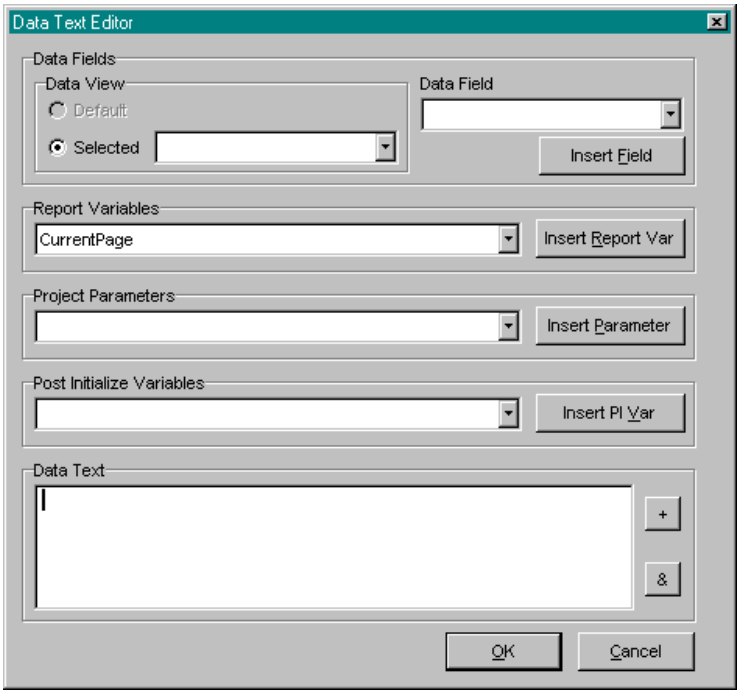
6. Del menú de Delphi seleccione la opción File | Use Unit y del cuadro de diálogo que se presenta seleccione la opción UDatamodule y pulse click en el botón OK.

Ahora procederemos a agregar los componentes Rave para crear los reportes.

7. Agregue al data module un componente TRvProject (pestaña Rave), modifique su propiedad Name al valor Rvproject1
8. A continuación agregue en el data module 2 componentes TRvTableConnection (pestaña Rave) y modifique sus propiedades como se muestra a continuación.

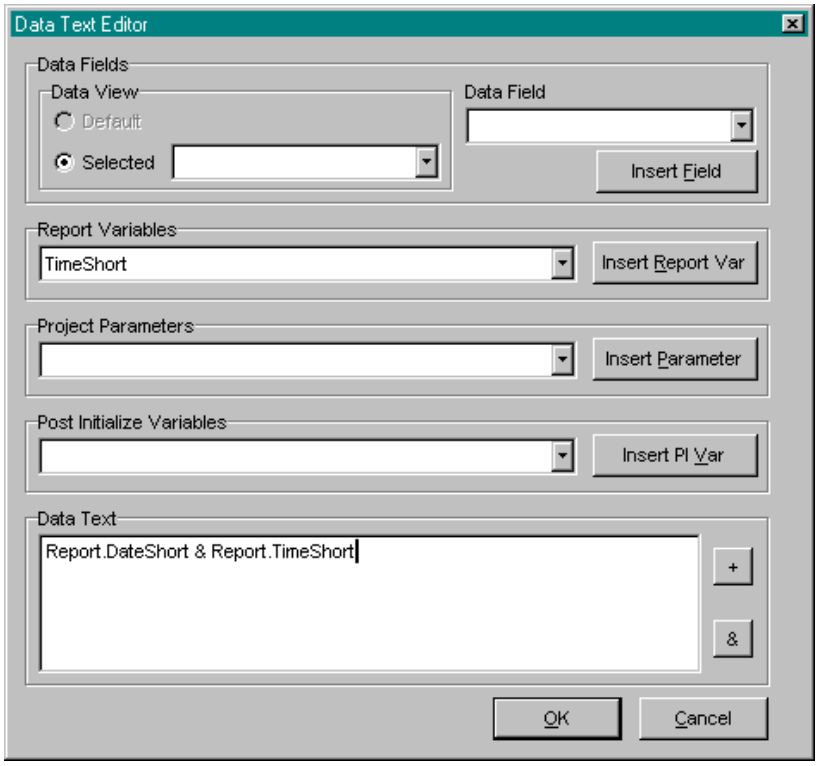
COMPONENTE	PROPIEDAD	VALOR
RvTableConnection1	Name Table	RvTableClientes Tclientes
RvTableConnection2	Name Tabla	RvTableabonos Tabonos

9. Pulse doble click sobre el componente Rvproject1 en el data module para abrir el diseñador visual de reportes.
10. Pulse click en la pestaña Report de la Paleta de Componentes del diseñador y coloque un componente Data Text en la esquina superior derecha de la página del reporte.
11. Del componente Data Text que acaba de colocar seleccione su propiedad DataField, en el editor de propiedades situado a la izquierda en el diseñador de reportes y pulse click en el botón con los tres puntos para invocar el cuadro de diálogo que se muestra a continuación:

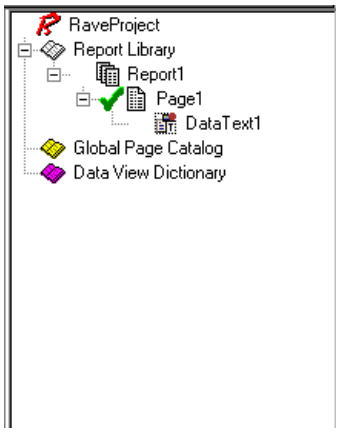


12. Dentro de este cuadro de diálogo, seleccione del combobox Report Variables la opción DateShort y pulse click en el botón insert Report Var, con esto estamos indicando que el componente Data Text mostrará la Fecha del sistema en formato corto.

13. Pulse click en el botón que se encuentra en la sección inferior del cuadro de diálogo, para agregar un espacio en blanco en el texto del componente Data Text. Seleccione del combobox Report Variables la opción TimeShort y pulse click en el botón Insert Report Var. Pulse click en el botón OK para cerrar el cuadro de diálogo.



14. Seleccione el componente Data Text y modifique su propiedad Font Justify al valor pJRight, con esto, la fecha y hora aparecerán en la esquina superior derecha de cada página de su reporte.
15. Pulse click en el signo + del elemento Report Library en el árbol del lado derecho del diseñador de reportes, continué expandiendo los nodos hasta que estén desplegados los elementos: Report1, Page1 y DataTextl.

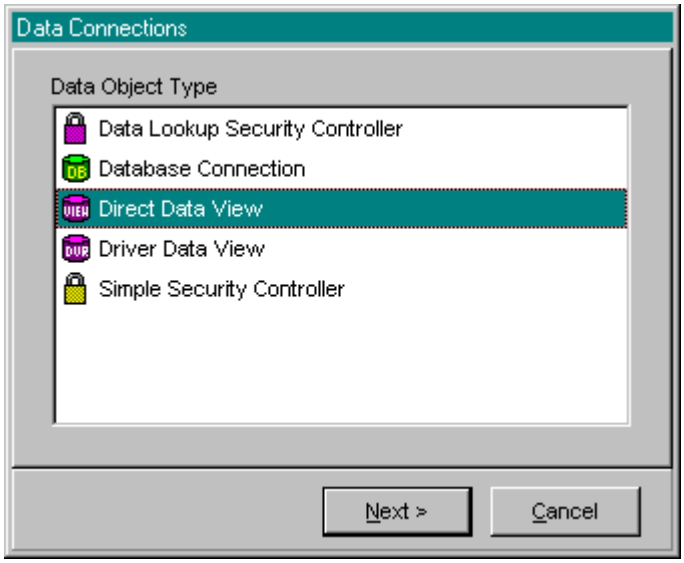


El árbol proporciona una manera fácil de visualizar todos los objetos del reporte y seleccionarlos cuando sea necesario.

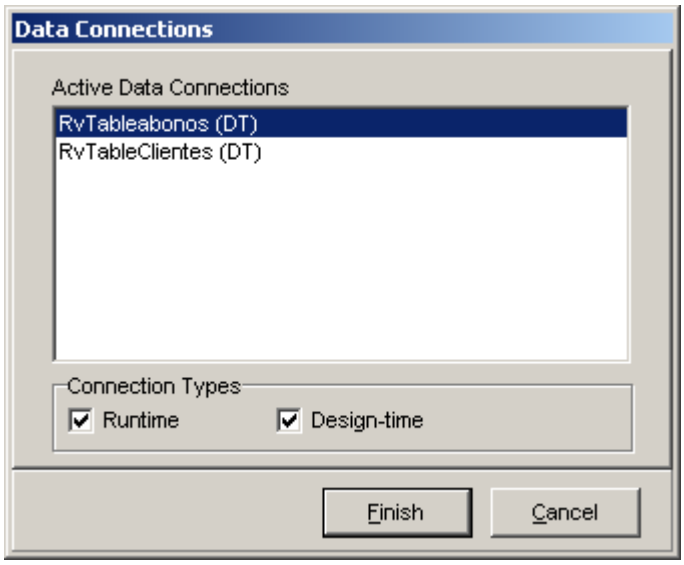
16. Desde la pestaña Standard coloque un componente Text en la página del reporte y posiciónelo cerca del margen superior izquierdo, modifique su propiedad Text al valor Reporte de Abonos x Cliente, y configure su propiedad Font en Arial, negrita de 12 pts.

Para utilizar la información de una base de datos en un reporte se debe crear un data view para cada componente dataset que provea información al reporte.

17. Pulse click en el botón New Data Object ubicado en la barra de herramientas del diseñador de reportes, en el cuadro de diálogo Data Connections seleccione la opción Direct Data View, tal como se muestra en la siguiente figura.



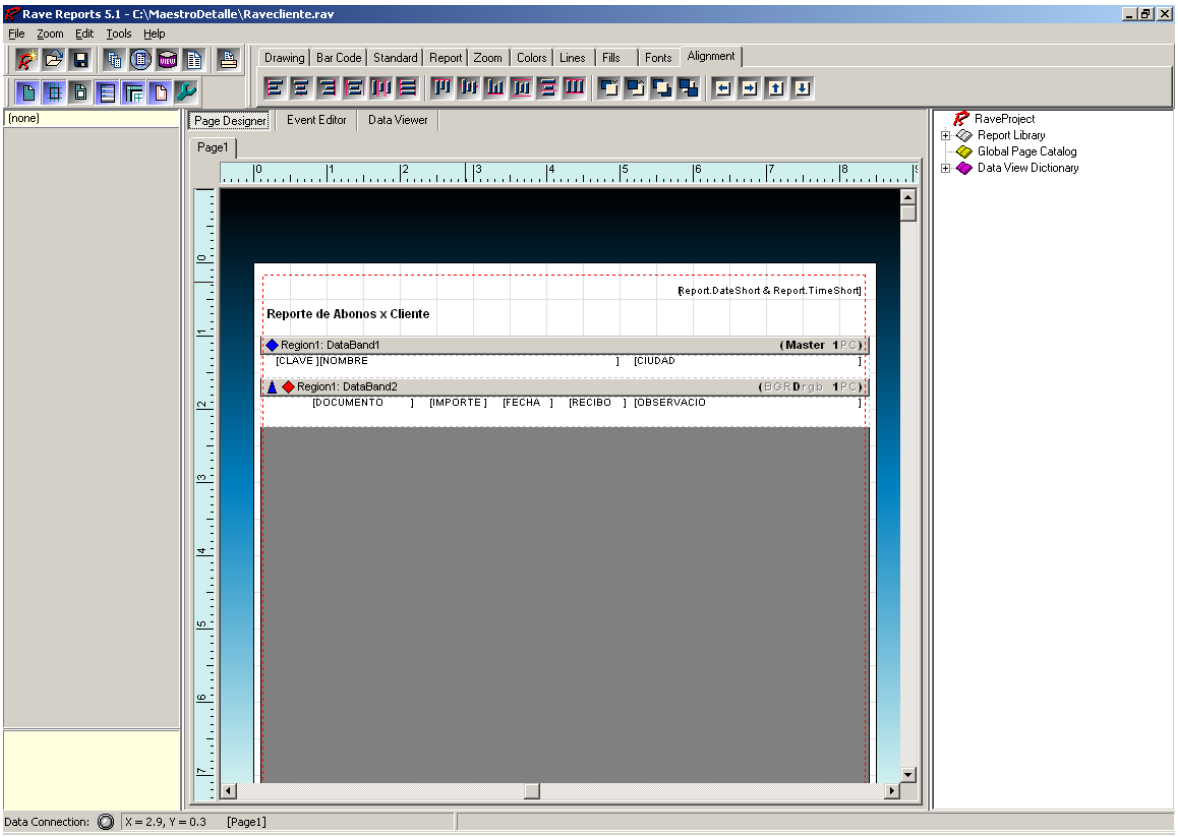
18. Pulse click en el botón Next, seleccione las conexiones RvTableabonos, RvTableclientes y pulse click en el botón Finish, como se muestra en la siguiente figura.



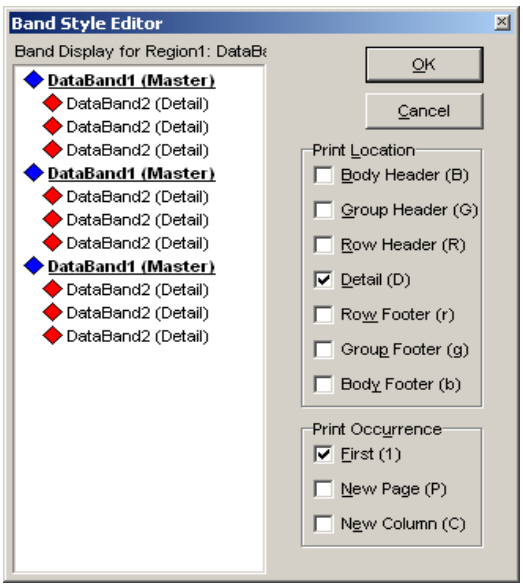
Agregando Bandas.

Antes de poder agregar a nuestro reporte los controles que desplegarán la información, debemos agregar algunos componentes de bandas.

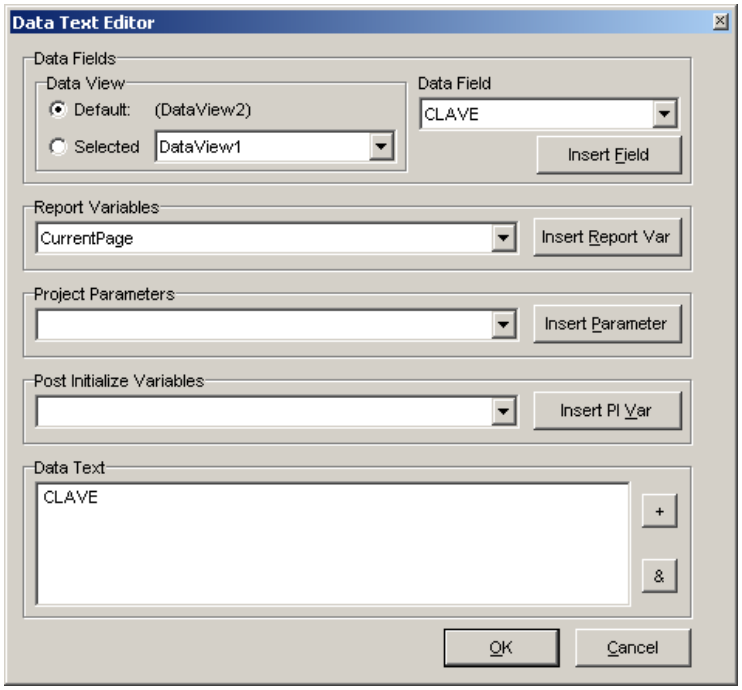
19. Las bandas deben de colocarse en un componente de Región, para eso vaya a la pestaña Report de la paleta de componentes del diseñador y agregue un componente Región a la página del reporte.
20. Ubique el componente Región debajo de los componentes Data Text y expándalo de tal manera que llene el espacio entre los márgenes izquierdo y derecho. Los márgenes están indicados por líneas punteadas en color rojo, podemos colocar tantos componentes Región en una página como deseemos.
21. Pulse click en el botón Save Project en la barra de herramientas del diseñador de reportes y grabe el proyecto con el nombre Ravecliente.rav, en el mismo directorio donde almaceno su proyecto Rave de Delphi.
22. De la pestaña Report agregue dos componentes Data Band dentro del componente Región, como se muestra en la siguiente figura.



23. Modifique la propiedad Name de la primer banda al valor Datosclientes y su propiedad DataView al valor Dataview1.
24. Pulse click en botón con los tres puntos de la propiedad BandStyIe de la banda Datosclientes para abrir el cuadro de diálogo Band Style Editor.



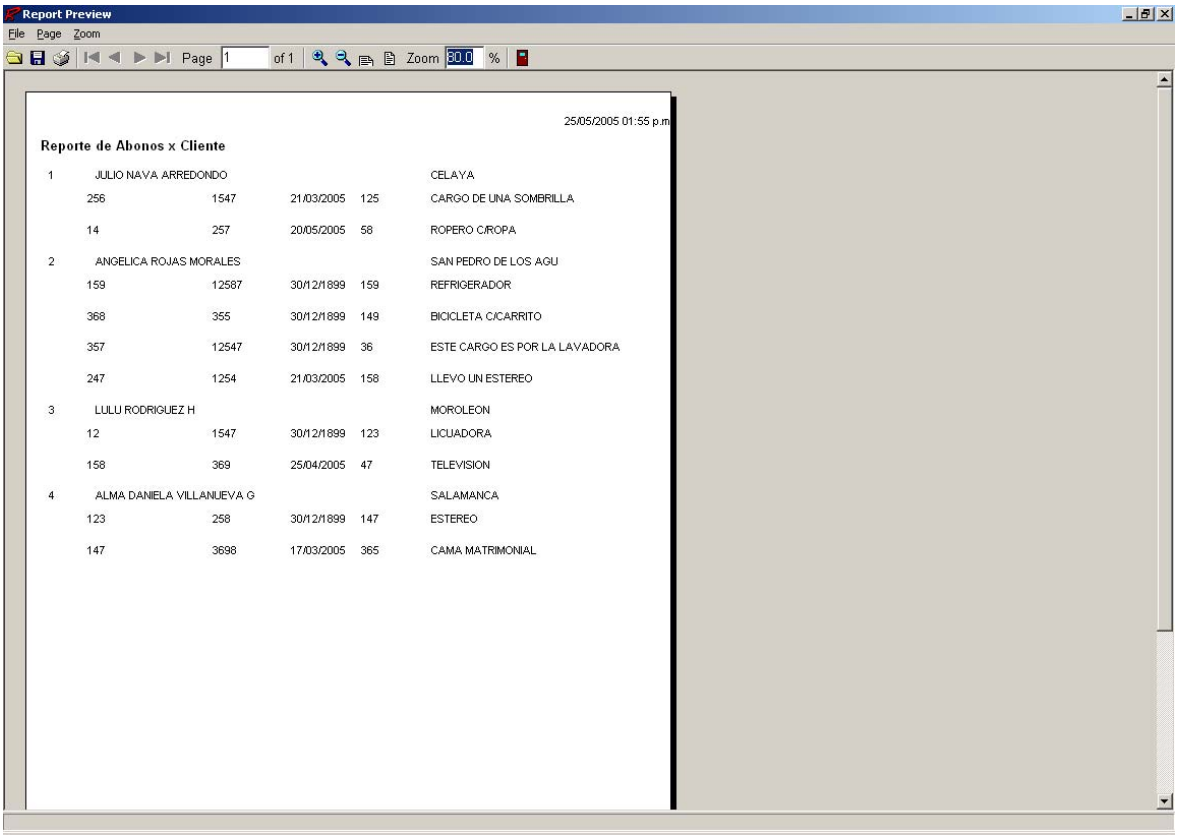
25. Seleccione la opción Detail dentro de la sección Print Location y pulse click en el botón OK.
26. Seleccione la segunda banda y modifique su propiedad Name al valor Datosabonos, su propiedad DataView al valor DataView2 y configure su propiedad BandStyle a Detail, por último, modifique su propiedad ControllerBand al valor Datosclientes.
27. Para ligar el data view de abonos al data view de clientes, coloque la propiedad DetailKey de la banda Datosabonos al valor Clave, para esto, pulse click en el botón de tres puntos de la propiedad DetailKey y del cuadro de diálogo que se presenta seleccione el campo clave dentro del listbox Data Field y pulse click en el botón insert Field.



28. Pulse click en el botón OK para aceptar la configuración del cuadro de diálogo.
29. Finalmente modifique de la banda Datosabonosl a propiedad MasterDataView al valor Dataview1, y su propiedad MasterKey a Clave.
30. El siguiente paso es agregar componentes Data Text a las bandas para desplegar los datos de las tablas clientes y abonos. A continuación, expanda el nodo Data View Dictionary en el árbol del diseñador y seleccione el nodo Dataview1. Para desplegar los campos de la tabla clientes.
31. Mantenga presionada la tecla Ctrl y arrastre el campo Clave a la banda CustomerDetail. Repita el mismo proceso para todo los campos que desee.
32. Expanda el nodo DataView2 en el árbol del diseñador y arrastre los campos necesariosa la banda Datosabonos.
33. Una ves terminado el reporte nos regresamos al proyecto en Delphi y le craemos su evento al boton Reporte General.

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
DATAMODULE2.RVPROJECT1.EXECUTE;
end;
```

34. Pulse la tecla F9 para visualizar su reporte.


Report Preview
File Page Zoom
Page 1 of 1
Zoom 80.0 %
25/05/2005 01:55 p.m.
Reporte de Abonos x Cliente
1 JULIO NAVA ARREDONDO CELAYA
256 1547 21/03/2005 125 CARGO DE UNA SOMBRILLA
14 257 20/05/2005 58 ROPERO C/ROPA
2 ANGELICA ROJAS MORALES SAN PEDRO DE LOS AGU
159 12587 30/12/1899 159 REFRIGERADOR
368 355 30/12/1899 149 BICICLETA C/CARRITO
357 12547 30/12/1899 36 ESTE CARGO ES POR LA LAVADORA
247 1254 21/03/2005 158 LLEVO UN ESTEREO
3 LULU RODRIGUEZ H MOROLEON
12 1547 30/12/1899 123 LICUADORA
158 369 25/04/2005 47 TELEVISION
4 ALMA DANIELA VILLANUEVA G SALAMANCA
123 258 30/12/1899 147 ESTEREO
147 3698 17/03/2005 365 CAMA MATRIMONIAL

11.4 Preguntas de Repaso

1. Describa la funcionalidad que ofrecen los componentes de Rave Reports.

2. Utilizando el programa de agenda que realizaste anteriormente crea un reporte donde muestre los nombre de contactos, telefono y ordenados por ciudad.

3. Para que sirve el componente Région dentro de los Reportes Rave.

4. ¿Qué caracteristicas tiene el diseñador visual de Rave Reports?

5. ¿Cuál es la funcion del componente RvProject?

6. ¿Cuál es la funcion del componente RvDataSetConnection?

7. Crea un reporte de maestro-detalle utilizando los archivos Clientes.db y abonos.db.

8. Crea un reporte donde obtengas la de facturación de una tienda con los campos cantidad, descripción y precio.

CONCLUSION

Después de haber estudiado, analizado y planteado el Tema "DESARROLLO DEL MANUAL TEORICO-PRACTICO BASADO EN DELPHI 7'', en este trabajo de Tesis he llegado a la conclusión de que es muy importante tener el conocimiento de varios lenguajes de programación los cuales nos ayudarán a tener una visión más amplia de lo que es en realidad la programación.

Mi finalidad para la elaboración de este manual es que sea de ayuda para el alumno ya que día con día las perspectivas de trabajo son más duras en la vida real, por lo cual tendrán más amplia su rama de trabajo, también nos ayudará a comprender algunas materias del plan de estudios de la carrera de Ingeniería en Computación.

El lenguaje Delphi me ha sido muy práctico en mi trabajo ya que tiene una gran amplia gama de utilerías y lo mejor que es basado a un lenguaje que ya había estudiado (pascal) por lo cual se me hizo más fácil la compresión y aprendizaje del lenguaje.

Durante la elaboración de mi tesis comprendí que no solo es programar ó comprender lenguajes, es más que eso, es administrar procesos para obtener la información requerida, es organizar parte de los recursos o procesos de la empresa, es administrar nuestro tiempo para atender cada una de las necesidades que van surgiendo dentro de la empresa por lo cual será mas productivo nuestro trabajo.

En el desarrollo de los módulos pude poner en práctica varias de los conceptos aprendidas durante la carrera, así como ampliar mis conocimientos sobre el manejo de bases de datos, organización de sistemas, etc.

Por medio de esta tesis aprendí a estructurar módulos de forma individual que trabajando conjuntamente formen un sistema sólido, a manejar un nuevo lenguaje, y programar de acuerdo a las necesidades del usuario. En mi tesis propongo ejemplos, explicaciones de cómo mejorar el trabajo dentro de una empresa que le serán de gran ayuda, también para el usuario final es muy importante tener la información en papel por eso el alumno también tendrá conocimientos en el Reporteador Rave.

Por último la elaboración de mi tesis de dio una visión muy amplia de lo que conlleva la programación de sistemas, también comprendí que no solo es necesario tener conocimientos de un solo lenguaje, por eso esta tesis es enfocada principalmente a los alumnos que sabrán utilizar cada una de las propiedades del lenguaje y desarrollar complejos sistemas que les abre gran oportunidad al mundo laboral, aunado con el aprendizaje del lenguaje Delphi

BIBLIOGRAFIA

- Marco Cantú
  MASTERING DELPHI 7
  Sybex
  Estados Unidos de America, 2003

- Ayuda del Software Delphi
- http://www.borland.com/delphi/
- http://www.clubdelphi.com/
- http://www.delphi.about.com
- http://www.delphi3000.com/
- http://www.delphimania.freeservers.com/DimeQueEs/N-Z.html
- http://www.marcocantu.com/EPascal/Spanish/ch02code.htm
- http://www.nevrona.com
- http://www.programacion.net/
- http://www.programacion.net/foros/28/msg/141755
- http://www.soodelphi.com/
- http://www.terra.es/personal/resfer/delphi/
- http://www.torry.net/