

GENERALITÀ SUI LINGUAGGI DI PROGRAMMAZIONE	2
COS'È DELPHI	4
AMBIENTE DI SVILUPPO INTEGRATO :	5
DRAG AND DROP DESIGN.....	5
STRUMENTI BIDIREZIONALI	5
COMPILATORE IN CODICE NATIVO	5
CONNESSIONE AI DATABASE INTEGRATA	5
CRONOLOGIA DEI PRODOTTI BORLAND.....	6
<i>Il linguaggio Pascal</i>	6
<i>Object Pascal : il Pascal secondo Borland...</i>	8
<i>Il nuovo modello di Oggetti</i>	9
<i>Le Proprietà</i>	10
<i>Perché Delphi si appoggia a Object Pascal?</i>	10
<i>Componenti - la via della riusabilità</i>	10
<i>Introduzione ai Componenti Visuali</i>	12
I PRINCIPALI COMPONENTI DELLA VCL DI DELPHI.....	14
GESTIONE DEL TESTO.....	14
<i>ListBox e ComboBox</i>	15
<i>RadioButton e CheckBox</i>	16
<i>Griglie</i>	16
<i>Button e SpeedButton</i>	17
<i>Notebook</i>	17
COMPONENTI DELPHI NON VISUALI	18
<i>I Menu</i>	19
<i>Menu e status-bar</i>	20
<i>Menu pop-up</i>	20
EVENTI PERIODICI: LA CLASSE TTIMER.....	20
<i>Common Dialog</i>	21
COMPONENTI DELPHI DEDICATI ALL'ACCESSO AI DATABASE	21
<i>I componenti dedicati all'accesso ai dati</i>	22
<i>TTable</i>	23
COMPONENTI "DATA-BOUND"	24
<i>Bibliografia Essenziale</i>	25

Introduzione al linguaggio visuale "Delphi"

prof. Cleto Azzani

Brescia luglio 2001

Generalità sui linguaggi di programmazione

La programmazione degli elaboratori può essere attuata facendo uso del linguaggio macchina oppure di linguaggi di programmazione detti ad alto livello.

Il linguaggio macchina è un linguaggio specifico di una determinata CPU e quindi presuppone una “perfetta conoscenza” del set di istruzioni del modo di indirizzamento, delle modalità di esecuzione delle istruzioni da parte della CPU e dell'architettura del sistema utilizzato.

I linguaggi ad “alto livello” tendono ad essere indipendenti dal tipo di CPU su cui operano, sono molto più aderenti al modo di esprimersi di un essere umano e non richiedono una conoscenza specifica dell'architettura del sistema su cui viene redatto un determinato programma.

Tra i linguaggi ad alto livello storicamente affermatasi ricordiamo :

- FORTRAN (FORmulae TRANslator) nato per la risoluzione di problemi di tipo scientifico.
- COBOL (COmmon Business Oriented Language) nato per la risoluzione di problematiche di tipo commerciale.
- BASIC (Beginner's All-purpose Symbolic Instruction Code) linguaggio adatto per un primo impatto con la realtà del calcolatore: risolve sia problemi di tipo scientifico che problemi di carattere commerciale; molto semplice da imparare ma non strutturato
- PASCAL linguaggio adatto a risolvere sia problemi di tipo scientifico che problemi di altro genere. Nato per l'insegnamento dei principi fondamentali della programmazione strutturata.
- C linguaggio nato con l'obiettivo di essere utilizzato in modo efficiente nella scrittura di sistemi operativi o di strumenti software;

Il programma scritto dal programmatore detto programma SORGENTE (SOURCE-PROGRAM); quello materialmente eseguito dalla CPU e perciò tradotto in linguaggio macchina viene denominato programma OGGETTO (OBJECT-PROGRAM). Si rende

necessario disporre di un traduttore che partendo da istruzioni redatte in linguaggio ad alto livello le trasformi in equivalenti istruzioni in linguaggio macchina.

Ogni linguaggio dispone quindi di un traduttore in linguaggio macchina.

Un generico linguaggio ad alto livello può essere strutturato per funzionare in modo INTERPRETE o in modo COMPILATORE.

Nel modo INTERPRETE il programma sorgente si trova in memoria RAM e viene introdotto attraverso una tastiera che opera in codice ASCII (American Standard Code for Information Interchange); quando viene attivata la esecuzione del programma l'interprete per ciascuna riga di programma effettua le seguenti operazioni:

- 1)- Controllo sintattico sulla riga di programma.*
- 2)- Traduzione della riga di programma in linguaggio macchina.*
- 3)- Esecuzione di quella riga di programma.*

Nel modo COMPILATORE il programma sorgente si trova di solito su disco sotto forma di "file" (archivio di informazioni su disco); la preparazione del sorgente su disco avviene attraverso l'uso di un apposito "strumento software" denominato EDITORE.

Il compilatore fa una analisi sintattica di tutto il programma sorgente fornendo eventuali segnalazioni di errore che servono al programmatore per correggere il programma. In caso il controllo sintattico abbia dato esito positivo, viene effettuata la traduzione in linguaggio macchina del programma sorgente, traduzione che viene registrata su disco sotto forma di "file binario".

Il programma oggetto così prodotto dal compilatore in genere non è ancora eseguibile in quanto privo delle necessarie informazioni base per eseguire operazioni matematiche più o meno complesse o operazioni di Input Output. Si rende necessaria un'ultima operazione denominata operazione di LINK delle librerie che viene attuata per mezzo di un altro "strumento software" chiamato LINKING-LOADER. Il programma oggetto prodotto dal compilatore, "linkato" con le routine di libreria è quindi in grado di funzionare autonomamente; anch'esso viene salvato su disco e costituisce il prodotto finale e conclusivo della compilazione.

Gli ambienti TURBO-PASCAL, TURBO-C, TURBO-BASIC della Borland; QUICK-C, QUICK-BASIC della Microsoft sono ambienti integrati in cui si può operare sia in modo Interprete che in modo Compilatore. In ambiente TURBO-PASCAL i "files-sorgenti" redatti in linguaggio PASCAL sono files di testo con estensione PAS per distinguerli dai "files-eseguibili" contenenti il codice macchina 8088-8086 che hanno estensione EXE .

Il FORTRAN è nato ed ha conservato la sua iniziale struttura di linguaggio Compilatore; il BASIC invece è nato come linguaggio Interprete e solo in tempi relativamente recenti sono state rilasciate edizioni di Compilatori BASIC .

Con lo sviluppo di CPU sempre più veloci e complesse, con l'abbandono da parte di Microsoft del DOS e con l'avvento e la diffusione di Windows 3.1 prima e successivamente di Windows 95 e poi 98 e contemporaneamente di NT, si è reso necessario cambiare l'approccio tradizionalmente usato nel mondo della programmazione sono così nati i linguaggi visuali fra i quali Visual Basic di Microsoft e Visual Pascal o meglio Delphi in casa Borland.

Cos'è Delphi

Delphi è un ambiente di programmazione visuale ad oggetti per lo sviluppo rapido di applicazioni (RAD / Rapid Application Development) a carattere generale e di applicazioni client/server per Windows 95 e 98 e Windows NT.

Con Delphi è possibile creare applicazioni Windows altamente efficienti riducendo al minimo i tempi di programmazione.

Delphi comprende una libreria di componenti riutilizzabili VCL e un insieme di strumenti di progettazione RAD, tra cui i modelli di applicazioni standard e di schede expert di programmazione. Con questi strumenti e con il compilatore Delphi a 32 bit è possibile creare rapidamente e testare prototipi, trasformandoli in robuste applicazioni perfettamente in linea con le moderne esigenze.

Delphi può essere utilizzato per sviluppare qualsiasi tipo di applicazione, dalle utility di analisi e test dei PC, fino ai più sofisticati strumenti di accesso ai database.

Gli strumenti di gestione dei database e i componenti di gestione dei dati previsti in Delphi permettono di sviluppare strumenti di gestione dati e applicazioni client/server in tempi notevolmente ridotti.

Con i controlli di gestione dei dati di Delphi, i dati vengono visualizzati direttamente durante la creazione dell'applicazione, consentendo una immediata verifica del risultato delle interrogazioni al database e delle modifiche all'interfaccia dell'applicazione.

Ambiente di sviluppo integrato :

L'ambiente di sviluppo integrato di Delphi IDE (Integrated, Development, Environment) mantiene sviluppo, verifica e gestione delle applicazioni in un unico ambiente. E' possibile creare o modificare una applicazione compreso schede di inserimento dati, report, menu, finestre di dialogo, database e definizioni di file, moduli dati, componenti, senza uscire da Delphi.

Drag and drop design

Delphi aumenta la produttività automatizzando le operazioni di programmazione ripetitive. E' possibile creare applicazioni semplicemente trascinando pochi componenti dalla Component Palette in una scheda chiamata form creando l'architettura dell'applicazione velocemente e facilmente, con il minimo di programmazione.

Strumenti bidirezionali

Procedendo con la selezione e la modifica delle proprietà dei componenti e delle schede, i risultati vengono aggiornati automaticamente nel codice sorgente, e viceversa. Modificando il codice sorgente direttamente con un editor di testo, come il Code Editor incorporato, le modifiche vengono immediatamente riflesse negli strumenti visuali.

Compilatore in codice nativo

Il compilatore Delphi 32-bit, sviluppato per ottimizzare il codice nativo produce applicazioni "self-contained", in codice nativo eseguibile .EXE. Oltre ad avere migliorato le prestazioni degli eseguibili compilati (in confronto ai linguaggi interpretati) la natura self-contained di un'applicazione Delphi consente l'indipendenza dalle librerie runtime .DLL.

Connessione ai database integrata

Il Borland Database Engine BDE, inserito in Delphi permette di sviluppare applicazioni in grado di accedere ai database Paradox, dBase, e InterBase. Attraverso SQL Links, l'applicazione può accedere anche a server SQL, tra cui InterBase, Oracle, Sybase, e Microsoft SQL. Inoltre, il flessibile motore di gestione di database di Delphi facilita il passaggio da applicazioni locali ad applicazioni client/server.

Cronologia dei prodotti Borland

La Borland dopo avere distribuito Turbo Pascal 6.0 per DOS ha introdotto sul mercato un prodotto per Windows 3.1 e lo ha chiamato Borland Pascal 7.0 (versione Windows di Turbo Pascal 7.0). Successivamente ha completamente rivisto la struttura dei suoi compilatori e strumenti di sviluppo accessori introducendo la famiglia Delphi.

A tutt'oggi sono state rilasciate ben 6 edizioni di Delphi :

Delphi 1 per lo sviluppo di applicativi a 16 bit in ambiente Windows 3.1 e 3.11

Nell'ordine si sono poi succedute le versioni di Delphi 2 , Delphi 3, Delphi 4, Delphi 5, Delphi 6 (da pochi giorni) per lo sviluppo di applicazioni a 32 bit per Windows 9x e Windows NT. I compilatori Borland sono generalmente disponibili in tre versioni :

- a) versione desktop (a basso costo)
- b) versione professional
- c) versione client/server o enterprise

La versione desktop pur costando poche centinaia di migliaia di lire è una versione completa ma priva di oggetti VCL particolari o di driver nativi per database particolari (opzioni richieste da un programmatore che sviluppi applicativi professionali) è comunque generalmente adatta per sviluppare applicazioni di tipo stand-alone.

Recentemente è stata introdotta sul mercato pure una versione di Delphi che opera in ambiente Linux (denominata Kylix) ciò consentirà di trasferire in questo nuovo ambiente, applicazioni native sviluppate per l'ambiente Windows.

Il linguaggio Pascal

Il linguaggio di programmazione di Delphi è basato sul Pascal, introdotto da Niklaus Wirth (università di Ginevra) appositamente come mezzo di insegnamento della programmazione strutturata. Per questo motivo tale linguaggio risulta essere assai elegante e leggibile, soprattutto se lo confrontiamo con linguaggi analoghi come C/C++, più strutturalmente vicino al linguaggio macchina oppure al Basic, che solo in tempi recenti ha subito una profonda riadattamento ai canoni della programmazione strutturata.

La facilità di lettura si accompagna simmetricamente alla facilità di scrittura: Se si tiene conto del fatto che Delphi è un compilatore velocissimo, scrivere programmi in Pascal per Windows significa, beneficiare di un ottimo compromesso tra velocità degli eseguibili prodotti e facilità di scrittura del codice.

Pascal è un linguaggio procedurale, dotato di *tipizzazione forte*. Questo sta a significare che per *ogni* variabile utilizzata all'interno di programmi Pascal deve essere *esplicitamente* dichiarato un *tipo* (ad esempio *integer*, variabile numerica intera; *real*, variabile numerica reale, ecc.). Ad ogni variabile, salvo qualche eccezione per i valori numerici, deve essere assegnato un valore del tipo corrispondente, pena errori in fase di compilazione. Senza questi vincoli, il compilatore non potrebbe risultare altamente ottimizzante e si correrebbe il rischio di errori a run-time dovuti ad assegnamenti "misti" non voluti dal programmatore (bug) e non intercettati dal compilatore che non sarebbe più in grado di effettuare controlli approfonditi sulla validità formale degli assegnamenti.

*Dalla versione 2 di Delphi è stato introdotto un nuovo tipo di dato, chiamato **Variant**, in grado di agire da contenitore per qualsiasi tipo di dato fondamentale di Delphi. Nel caso si utilizzino variabili di tale tipo, non è possibile ottimizzare il codice e catturare a tempo di compilazione errori nella codifica. Se ad esempio, una variabile di tipo Variant contiene una stringa ed un'altra un numero intero, il compilatore non segnala errore se applico ad esse l'operatore di somma. Otterrò un errore in fase di esecuzione ("run-time"), certamente più difficile da scoprire.*

Il linguaggio Pascal rende disponibile al programmatore un set completo di tipi di dato di base, che spaziano dai numeri interi (*integer*) fino ad arrivare a stringhe (*string*) la cui gestione in memoria è trasparente al programmatore, passando per il tipo *boolean*, che è in grado di esprimere i valori di verità *true* e *false*, definiti come costanti *del linguaggio*.

Viene offerta anche una completa gestione di puntatori ed allocazione dinamica di memoria, cosa che permette di creare a run-time nuove entità le cui dimensioni non devono essere decise a tempo di codifica. Questo si traduce in un'ottimizzazione dello sfruttamento delle risorse della macchina su cui si sta lavorando. Combinando i meccanismi di allocazione dinamica alla possibilità di derivazione di nuovi tipi di dato strutturati a partire dal costrutto *record*, inoltre, è possibile costruire e gestire con relativa facilità tipi di dato ricorsivi come "liste" e gli "alberi".

Altra caratteristica importante del linguaggio Pascal è quella di essere *case-insensitive*, ovvero il compilatore non distinguerà in alcun punto del programma (tranne che all'interno

delle stringhe...) tra lettere maiuscole e minuscole. Saranno quindi considerati identici nomi di variabili e funzioni come *MiaVariabile* e *miavariabile*, a differenza di quanto accade in linguaggi come C/C++, ed in analogia con il Basic.

L'operatore di assegnamento utilizzato dal Pascal è :=, a differenza della stragrande maggioranza dei linguaggi procedurali. Pur essendo più corretto dal punto di vista matematico, questo potrebbe essere disorientante per chi proviene da altri linguaggi.

E' possibile inglobare blocchi di codice all'interno delle strutture di controllo, usando la keyword di apertura *begin* e quella di chiusura *end*.

Ogni istruzione deve essere terminata da un punto e virgola “;”.

I commenti sono porzioni dei sorgenti racchiusi tra parentesi graffe { *commento* } oppure dalla combinazione di parentesi tonda ed asterisco (* *commento* *)

Pascal distingue tra *funzioni* e *procedure*: le prime, dichiarate utilizzando la keyword *function*, restituiscono un valore, mentre le seconde no. Ad esempio, le linee di sorgente appena mostrate definiscono il codice di una procedura (si noti la keyword *procedure* prima della definizione del nome).

Le variabili locali dichiarate all'interno di procedure oppure quelle *globali* (visibili da tutte le procedure appartenenti al programma) devono essere dichiarate all'interno di sezioni separate, diversamente da quanto previsto in altri linguaggi (esempio, in C/C++).

Object Pascal : il Pascal secondo Borland...

Fin dai tempi del Turbo Pascal, Borland ha proposto estensioni al Pascal standard. A parte l'introduzione delle caratteristiche object-oriented, una delle più importanti è stata quella delle *unit*, costruito che permette una programmazione *modulare*.

Attraverso la struttura delle *unit* è possibile creare programmi accostando diversi moduli di codice visibili solo tramite un'interfaccia data dalle dichiarazioni delle funzioni in esse contenute.

Con le *unit* Borland fornisce la possibilità di costruire entità contenenti codice nelle quali si può restringere l'accesso dall'esterno a particolari funzioni dichiarate in una sezione di *interfaccia*.

Le unit sono molto utilizzate in Delphi, visto che ad ogni finestra del proprio programma viene associata una unit che ne contiene la definizione.

Rispetto alle versioni precedenti dei compilatori Pascal di Borland, Delphi introduce alcune interessanti innovazioni, tra le quali spiccano un *nuovo modello di oggetti*, la *variabile Result* di ritorno di una funzione ecc. ecc.

La nuova variabile Result permette di restituire oggetti o record tramite funzioni

Per quanto riguarda le innovazioni nella restituzione di valori nelle funzioni, ora con Delphi è possibile restituire *qualsiasi* tipo di dato, che sia semplice o strutturato, con poche eccezioni. All'interno delle funzioni, per specificare il valore da restituire, si fa riferimento alla variabile speciale Result, mediante la quale si è in grado di inizializzare differenti componenti di eventuali tipi strutturati.

Il nuovo modello di Oggetti.

Arriviamo ora alla novità più importante: Delphi introduce un nuovo modello di oggetti. Innanzi tutto sparisce la parola chiave *object* dalla definizione di classe, per far posto alla più usuale *class* (anche se la prima viene mantenuta per compatibilità verso l'indietro). Delphi ora permette di definire parti *protette*, ovvero visibili dai soli discendenti della classe (oltre che dall'interno della classe stessa). Questo in aggiunta alle parti pubbliche, visibili dall'esterno della classe, e private, invisibili totalmente dall'esterno, e accessibili quindi dalle sole funzioni appartenenti alla classe.

Ora *ogni* classe deriva da un unico super-oggetto, chiamato *TObject*. Questo è un oggetto astratto, nel senso che non può essere creata un'istanza di *TObject*, anche se ci si può riferire ad esso come parametro di funzione se si vuole che ad una funzione possa essere passato *qualsiasi* oggetto.

Le Proprietà.

I componenti che si disegnano sul video nell'ambiente di sviluppo hanno una serie di *proprietà* che determinano il loro comportamento a run-time. Le proprietà sembrerebbero coincidere, quali contenitori di dati legati ad un'istanza di classe, con le variabili membro. Internamente, però, a livello di codifica, offrono un meccanismo di gestione differenziata delle operazioni di lettura e scrittura. Una proprietà, infatti, può essere collegata a due funzioni a seconda che ne venga letto il valore (operazione di *read*) oppure ne venga aggiornato il contenuto (operazione di *write*).

Perché Delphi si appoggia a Object Pascal?

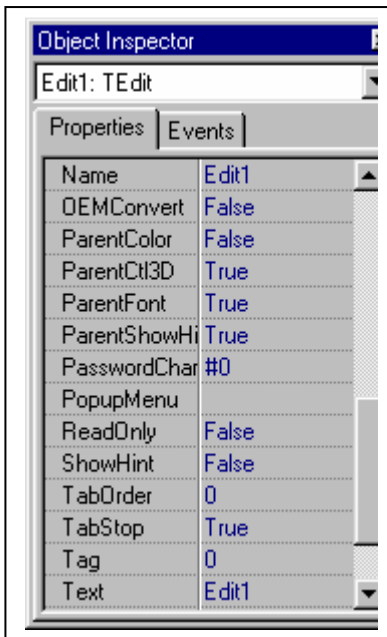
Molta della semplicità di scrittura e manutenzione di codice Object Pascal è dovuta alla possibilità, da parte dei progettisti Borland, di inventare e sfruttare estensioni proprietarie del linguaggio: è un dato di fatto, ormai, che Pascal è uno standard Borland e che quindi quello che Borland decide diventa standard de facto. Questo non può accadere con C++, dove ad occuparsi delle caratteristiche del linguaggio è un comitato di standardizzazione, che sicuramente non sta a modificare le caratteristiche di un linguaggio solo perché in ambiente DOS/Windows c'è qualcuno che potrebbe trarre vantaggi nella costruzione di un tool di programmazione visuale!

Componenti - la via della riusabilità.

Delphi risulta essere un ottimo tool di creazione di programmi che richiedono la gestione di interfacce con l'utente assai complesse. Questo è principalmente dovuto alla capacità di poter disporre in modo visuale un gran numero di elementi di interfaccia all'interno di finestre e dialog-box. Guardando all'interno della *component palette*



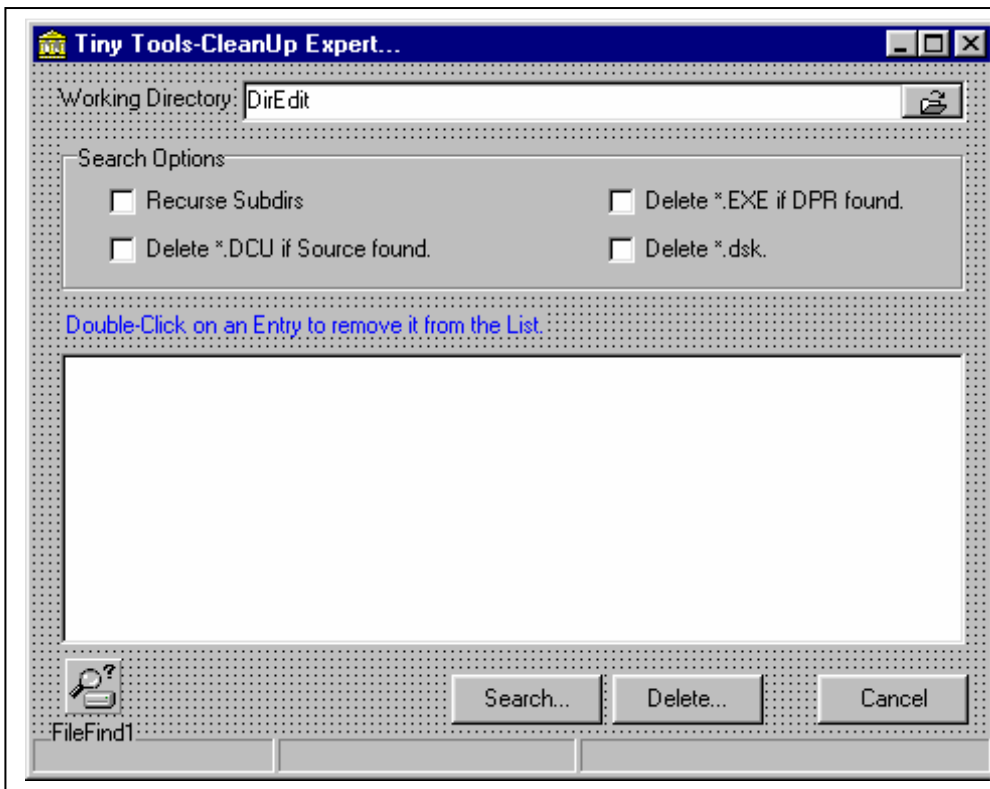
subito sotto alla linea del menu, nella parte destra dello schermo, si possono trovare praticamente tutti gli elementi di interfaccia cui siamo abituati utilizzando Windows: si va dai più usuali bottoni, edit-box e list-box, fino ad arrivare a sofisticati controlli tipo *tabbed notebook*, passando dagli *speedbutton* (bottoni generalmente utilizzati con sole bitmap, in grado di gestire comportamenti di mutua esclusione tipici delle toolbar) e dalle griglie di stringhe (*TStringGrid*). Tutti questi componenti possono essere piazzati con un semplice click all'interno dei nostri programmi. Una volta collocati sulla form, è possibile selezionarli e cambiarne le proprietà a design-time, attraverso l'*Object Inspector*, mostrato di seguito.



E' così possibile definire le scelte possibili in un gruppo di radio button, il tipo di font in una edit-box o il colore di una linea di testo statico (*TLabel*), il tutto senza passare attraverso la scrittura di alcuna riga di codice!

Mano a mano che si aggiungono componenti all'interno di una form, Delphi tiene aggiornati i sorgenti della propria applicazione. Di seguito viene mostrata una form piena di componenti

Se andiamo ad osservare i sorgenti creati automaticamente in risposta alle azioni che hanno portato a realizzare tale form,



(premendo F12 in modalità IDE) abbiamo un'idea della mole di codice che Delphi ci ha risparmiato di scrivere e, soprattutto, di mantenere ad ogni aggiornamento degli elementi di interfaccia .

Ognuno dei componenti forniti con Delphi altro

non è che una classe Object Pascal, scritta rispettando alcune regole di scrittura dei componenti e, generalmente, derivata da un tipo di dato presente in Delphi che permette di ereditare i comportamenti tipici di ogni componente.

Diversamente da quanto accade con altri ambienti di sviluppo visuali in ambiente Windows, come ad esempio Visual Basic, la scrittura dei componenti in Delphi passa attraverso Delphi stesso! Questo permette di raggiungere un duplice obiettivo: primo, non è necessario conoscere altri linguaggi per essere in grado di estendere l'ambiente di sviluppo con nuovi componenti scritti ad hoc; secondo, è possibile seguire, nello sviluppo

di nuovi componenti, una tecnica incrementale. Nel momento in cui ci si accorge che una serie di modifiche ad elementi di interfaccia già esistenti hanno portato alla creazione *de facto* di un nuovo elemento di interfaccia, è possibile promuovere questo elemento, con un minimo sforzo di codifica, a far parte della *Component Palette*, pronto per essere riutilizzato in altri progetti.

Introduzione ai Componenti Visuali

I componenti di Delphi si possono suddividere in tre grandi gruppi, più precisamente ***Visual, Non-Visual e Data-Bound Components***.

Andiamo per ordine e cominciamo con i primi: come si può capire dalla denominazione, si tratta di componenti che hanno la caratteristica di apparire a design-time (fase di progetto) con lo stesso aspetto con cui appariranno a run-time.

Il primo gruppo dei componenti *Visual* è quello dei controlli dedicati alla gestione del testo, attraverso i quali è possibile interagire con l'utente per mezzo della tastiera o, comunque, visualizzare messaggi sotto forma testuale.

Per primo incontriamo *TLabel*, attraverso il quale è possibile piazzare all'interno delle finestre una o più linee di testo. Si tratta di un componente atto solo alla *visualizzazione*, non quindi all'accettazione di input da tastiera. Al contrario, *TEdit* permette all'utente di inserire testo, dopo aver selezionato il componente con il mouse, oppure esserci spostati con il tasto Tab. Di *TEdit* esiste una versione specializzata per l'input formattato, che prende il nome di *TMaskEdit*. Entrambi accettano una sola linea di input, diversamente da quanto accade per il componente *TMemo*.

Chiudono l'elenco dei controlli legati al testo *TComboBox* e *TListBox*, metafore di combo-box e list-box, rispettivamente, le cui funzionalità dovrebbero essere note a tutti gli utilizzatori di Windows.

Passando ai bottoni, tre sono le classi offerte da Delphi che impersonano differenti tipi di pulsanti: *TButton*, *TBitBtn* e *TSpeedButton*. Il primo è il classico bottone di Windows, senza alcun elemento grafico - solo testo e bordo, che può essere scelto con lo stile di Windows 3.1 oppure di Windows 95!. *TBitBtn*, a differenza del precedente, può includere un *glyph* (piccolo elemento grafico) al suo interno, in modo da poter identificare più rapidamente l'azione associata. L'ultima classe, invece, è metafora di pulsanti appositamente creati per poter lavorare in gruppo ed essere dotati di *sola grafica*, adatti quindi a creare *toolbar*.

TCheckBox e *TRadioButton*, se qualcuno non l'avesse ancora capito dal nome, impersonano rispettivamente check-box e radio-button.

Il componente *TStringGrid*, permette di gestire una griglia di stringhe, con una serie di proprietà assai complete che vanno dalla possibilità di tenere fisse ed evidenziate un numero a piacere di colonne e righe, fino alla definizione del numero di righe e colonne nella griglia e una gestione automatica delle scroll-bar nel caso l'area di visualizzazione fosse insufficiente. Questo componente presenta una miriade di usi possibili: dalla possibilità di farne la base di uno spreadsheet, fino alla gestione delle tessere del gioco del quindici, o dei numeri di un tastierino numerico! Tale componente supporta anche funzioni di data-entry e di navigazione all'interno della griglia utilizzando il tasto Tab. Ovviamente tutti questi comportamenti sono parametrizzati e liberamente modificabili a design-time.

TGroupBox e *TPanel* permettono di raggruppare visivamente altri componenti, attraverso la visualizzazione di bordi rettangolari con titoli oppure riquadri con effetti tridimensionali. Se utilizzati con intelligenza, questi elementi di interfaccia permettono di migliorare sostanzialmente la leggibilità dei dati presenti nelle form delle proprie applicazioni.

Con *TRadioGroup* è possibile costruire una combinazione di *TGroupBox* (riquadro con titolo) ed una serie di radio-button. La gestione delle relazioni di mutua esclusione fra i bottoni è delegata al componente. Alla selezione di uno dei bottoni nel gruppo corrisponde un annullamento automatico della precedente scelta. E' possibile raggruppare i bottoni su una o più colonne, impostando semplicemente il valore di una proprietà (*Columns*).

TTabSet e *TNoteBook* permettono di costruire in modo assai semplice ed immediato tutta quella serie di controlli che simulano il comportamento dei catalogatori, costituiti da una serie di pagine sovrapposte e selezionabili attraverso il clic del mouse su un'area sempre visibile contenente il titolo della pagina. In alternativa è possibile utilizzare *TTabbedNotebook*, un controllo che in pratica offre i primi due già combinati.

Con *TImage* è possibile visualizzare immagini *bmp*, *wmf* ed *ico* all'interno di una porzione rettangolare della propria finestra, semplicemente impostando una proprietà con il nome del file appropriato. Attraverso *TPaintBox* si è in grado di offrire al proprio programma un'area rettangolare limitata su cui effettuare operazioni grafiche. *TShape*, invece, permette di inserire nelle proprie form, a design-time, alcuni elementi grafici, come ellissi, cerchi e rettangoli, con motivi e colori di riempimento e bordo definibili a piacere.

Tornando alle griglie, troviamo altri due interessanti componenti: *TColorGrid* e *TDrawGrid*. Il primo permette di far selezionare all'utente colori con combinazione di background e

foreground (ad esempio per offrire la possibilità di impostare a run-time i colori degli elementi della nostra applicazione). Il secondo, invece, è una griglia entro la quale è possibile effettuare operazioni grafiche.

Utilizzando *TGauge* è possibile visualizzare graficamente lo stato di avanzamento di elaborazioni dalla durata non breve. *TSpinEdit* e *TSpinButton* permettono di creare controlli di tipo *spin*, con frecce che permettono di incrementare e decrementare valori numerici: il primo, diversamente da *TSpinButton*, è collegato ad un campo di testo aggiornabile nel modo usuale, ovvero con mouse e tastiera.

Infine, molto utili in alcune applicazioni si potrebbero rivelare *TCalendar*, un calendario mensile con indicazione del giorno della settimana, e *TDirectoryOutline*, componente che effettua una rappresentazione ad albero della struttura delle directory.

I principali componenti della VCL di Delphi

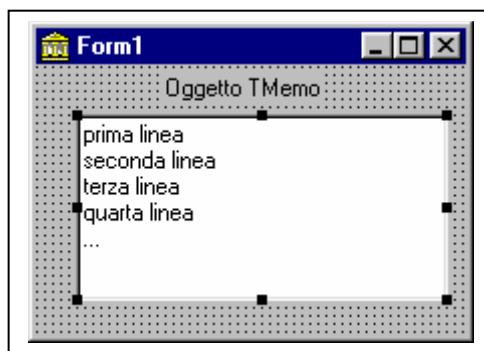
E' utile effettuare una suddivisione all'interno di tale gruppo di componenti, fra controlli dedicati alla *gestione del testo*, *bottoni*, *componenti grafici*, *griglie* e *componenti di accesso a directory*.

Gestione del testo.

A questo gruppo appartengono tutti quegli elementi di interfaccia in grado di gestire informazioni sotto forma testuale, sia che si tratti di semplice visualizzazione di stringhe di caratteri, oppure dell'insieme di funzionalità offerte da un text editor.



I componenti fondamentali per la gestione del testo sono *TLabel* e *TEdit*, rispettivamente in grado di gestire output ed input/output di righe di testo.



Conclude la rassegna dedicata ai componenti relativi alla gestione del testo la classe *TMemo*, in grado di manipolare input multilinea

Se il testo in un componente di tipo *TLabel* è contenuto all'interno della proprietà *caption* e per uno di tipo *TEdit* è contenuto nella proprietà *text*, entrambe di tipo *string*, per gli oggetti appartenenti a *TMemo* si

ricorre ad una struttura dati differente, in quanto si deve gestire un campo multilinea. La struttura dati utilizzata per la memorizzazione è un *array di stringhe*, appartenente alla classe *TStrings*.

All'interno della documentazione Borland si fa riferimento a tale oggetto chiamandolo *lista di stringhe*. L'accesso alle singole stringhe ivi contenute avviene allo stesso modo di un array, ovvero specificando il valore di un indice, mi sembra più corretto riferirsi alle *TStrings* chiamandole *array* (anche se non hanno un dimensionamento definito a compile-time, come i normali array di Object Pascal). La variabile membro di tipo *TStrings* che esprime il contenuto di un campo memo prende il nome di *Lines*. Le operazioni di base per agire sul contenuto di un campo *TMemo* sono sostanzialmente tre e fanno riferimento tutte ad operazioni sulle *TStrings*. Più precisamente si tratta di *add* - aggiunta di una stringa in fondo all'array; *delete* - operazione che cancella una particolare riga e *clear* - operazione in grado di cancellare l'intero contenuto del campo memo.

ListBox e ComboBox.



Anche se relative alla manipolazione di testo, le classi che rappresentano list-box e combo-box vanno considerate un po' a parte, dato che generalmente questi controlli vengono utilizzati nella loro modalità *read-only*, per mettere l'utente in condizione di poter scegliere fra una serie di alternative.

Le stringhe contenute in tali elementi possono essere definite sia a design-time che a run-time, agendo sulla proprietà *Items*, che è di tipo *TStrings* (similmente a quanto accadeva con la proprietà *Lines* degli oggetti *TMemo*). Ovviamente su questa proprietà è possibile applicare i metodi visti per i campi memo (*add*, *insert*, *clear* ecc.).

Nel nostro esempio, essendo l'insieme di stringhe noto a priori e fisso, questo è stato inserito a design-time utilizzando il property editor custom per liste di stringhe

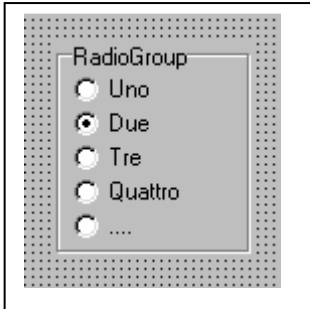


Per quanto riguarda la list-box, non è necessario impostare particolari proprietà in quanto, per definizione, una list-box è *read-only*, mentre una combo-box potrebbe avere input da tastiera da parte dell'utente.

Per rilevare i cambiamenti selezionati dall'utente a run-time, al solito si imposta la risposta all'evento *OnChange* dei due elementi di interfaccia. All'interno del codice di risposta si va a rilevare l'indice di linea evidenziato dall'utente, utilizzando la proprietà *ItemIndex*.

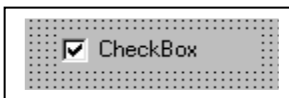
RadioButton e CheckBox.

Le classi *TRadioButton* o, meglio, *TRadioGroup*, e *TCheckBox*, sono gli oggetti che rappresentano un gruppo di radio button e di check-box, si prestano per consentire la scelta di opzioni all'interno di un programma.



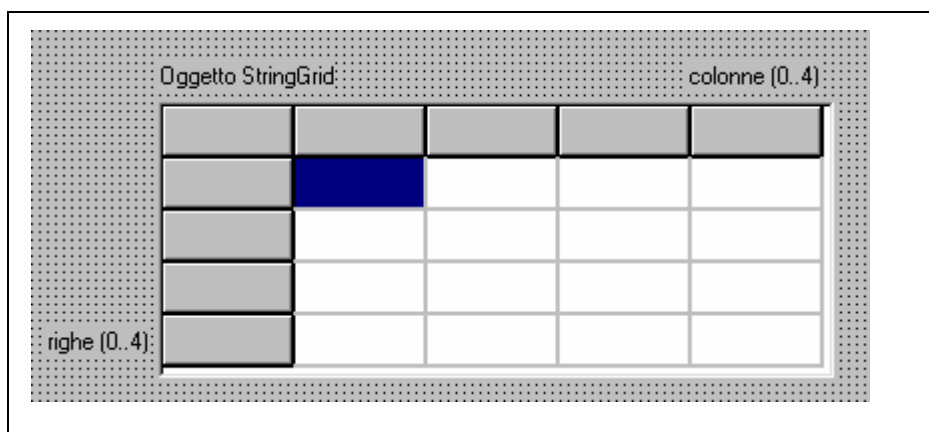
Il gruppo di radio button esprime sempre una serie di scelte in alternativa fra loro; mentre nel caso della *CheckBox* le scelte non sono più mutualmente esclusive.

La funzione di risposta attraverso la quale è possibile rilevare l'evento di cambiamento dello stato degli oggetti di interfaccia non è più di tipo *OnChange*, bensì *OnClick*. proprietà chiave!



Griglie

Una grande famiglia di componenti di Delphi, sia per numero che per importanza, è quella delle griglie: la prima è l'oggetto *TStringGrid*, che servono da base per tutta quella serie di applicazioni che necessitano di input/output di dati in formato tabellare.



TStringGrid, che altro non è che un array bidimensionale di stringhe, con righe e colonne numerate a partire dallo zero.

Una proprietà chiave da impostare a design-time è quella del numero di righe e colonne di titolo da tenere fisse (e che verranno evidenziate con l'effetto tridimensionale citato sopra), rispettivamente denominate *fixedCols* e *fixedRow*. Nell'esempio sono impostate entrambe al valore 1.

Appartenente alla famiglia delle stringhe, ed ai componenti è anche la classe *TDrawGrid* che, a differenza di *TStringGrid* che permette di memorizzare all'interno delle celle solo testo, è in grado di gestire in generale informazioni di tipo grafico.

Button e SpeedButton.



Tra i componenti visuali non potevano mancare i bottoni, elementi principe dell'interazione uomo/macchina nelle moderne interfacce GUI (Graphic User Interface).

In Delphi i bottoni "classici", quelli per intenderci senza grafica e con un solo stato (non possono, ad esempio, risultare Fondamentalmente l'interazione del programmatore con oggetti appartenenti a questa classe si limita, oltre alla definizione di elementi a design-time come *caption*, dimensioni e posizione, alla

risposta all'evento *OnClick*.

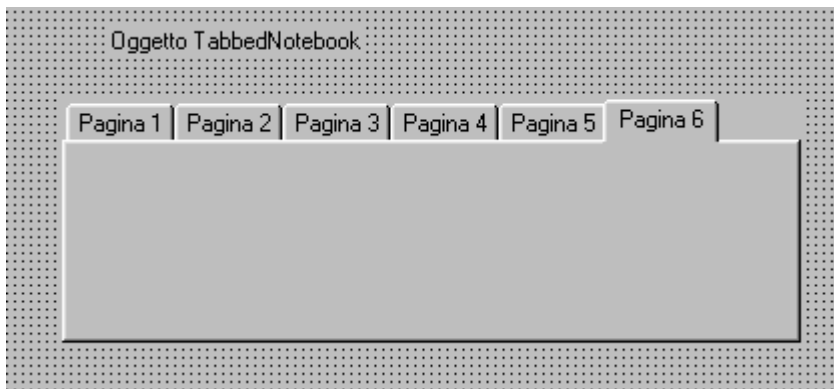
Esistono altre due classi di pulsanti in Delphi, una, chiamata *TBitBtn* (da **bit**map **but**ton), che in pratica aggiunge ai semplici bottoni la possibilità di inserire, oltre alla *caption* (la "scritta") anche un *glyph*, un elemento grafico in grado di far riconoscere al primo colpo d'occhio le funzionalità associate all'elemento.

Le particolarità di questo tipo di pulsanti, comunque, sono principalmente legate all'introduzione della grafica: per quanto riguarda le modalità di utilizzo, risultano perfettamente analoghe a quelle dei "semplici" *TButton*.

Diversamente, la terza classe di pulsanti, *TSpeedButton*, oltre ad essere personalizzabile con elemento grafico deciso dall'utente, ha la peculiarità di essere gestita *in gruppo* per creare toolbar, toolbox e comunque permette di gestire due stati (bottone premuto, bottone alzato), cosa che amplia enormemente gli ambiti di applicazione.

Notebook

L'insieme di controlli resi disponibili da Delphi è veramente ampio, tanto da includere due versioni distinte dello stesso controllo, ovvero gli ormai classici *notebook*. Tre sono le classi deputate alla gestione di tale elemento di interfaccia: *TTabSet*, *TNotebook* e *TTabbedNotebook*. Ad essere precisi le prime due non gestiscono un notebook completo, al contrario di quanto accade con la seconda.



Componenti Delphi non visuali

Molti degli oggetti della VCL appartengono alla categoria di elementi che hanno aspetto a run-time profondamente differente da quello a design-time. In generale questi componenti da soli non portano a termine compiti particolari. Rimangono invisibili e collegati alla form che li contiene, ma senza alcuna codifica aggiuntiva non hanno alcun effetto sull'interfaccia con l'utente.

I Componenti non visuali rappresentano una serie di elementi di interfaccia utilissimi nello sviluppo di applicazioni. Si va dai menu, sia di tipo tradizionale che di tipo "pop-up", passando dai timer in grado di far effettuare alle nostre applicazioni operazioni periodiche ed asincrone rispetto al normale flusso di eventi gestito dal programma.

Per la maggior parte degli ambiti di utilizzo, lo sfruttamento di tali componenti da parte del programmatore avviene in modo tradizionale, invocando metodi ad essi associati e modificandone le proprietà a run-time, oppure scrivendo routine di gestione degli eventi. Alcuni di questi componenti hanno a disposizione *custom property editor* per poter impostare le proprietà di default.

La classe *TMainMenu*, per esempio, offre un *menu builder* di tutto rispetto, ma neppure in questo caso il programmatore può esimersi dallo scrivere, in ultima istanza, codice di collegamento con altri oggetti scrivendo nei sorgenti "alla vecchia maniera".

Attraverso *TPanel*, infatti, osserveremo le fasi di creazione e gestione di una status-bar collegata alle informazioni della proprietà *Hint* delle singole voci di menu.

Grazie all'estrema ortogonalità della libreria di classi di Delphi ci si renderà conto di come una serie di problemi concettualmente simili (dalla visualizzazione degli *Hint* nella status bar, alla gestione contemporanea di voci di menu pop-up e normali) vengano risolti con blocchi costitutivi comuni, fatto che semplifica di molto le attività di programmazione.

I Menu.



Aggiungere menu alle proprie applicazioni è cosa assai immediata e passa attraverso l'utilizzazione di oggetti della classe *TMainMenu*, che si possono trovare all'interno della *component palette* nella sezione *Standard*.

Così come visivamente ogni menu è formato da voci, raggruppate eventualmente a formare sottomenu, un oggetto appartenente alla classe *TMainMenu* contiene una lista di voci di menu, rappresentate da oggetti *TMenuItem*.

Il processo di creazione di un menu si svolge mediante un *custom property editor*, chiamato *Menu Designer*.

E' sufficiente un doppio click sulla proprietà *items* per invocare tale editor di menu, attraverso il quale è possibile inserire nuove voci, spostare ed eliminare voci già esistenti e costruire visivamente gerarchie di sottomenu.

Ogniqualevolta si crea una nuova voce, all'interno del sorgente viene creato un nuovo oggetto di tipo *TMenuItem*, con le seguenti proprietà chiave:

- **Caption** - stringa contenente il testo visualizzato nella voce di menu. E' possibile definire acceleratori facendo precedere dal carattere "&" una delle lettere della stringa, alla pari di quanto mostrato per i componenti di tipo *TLabel* nella puntata precedente.
- **Checked** - proprietà di tipo *boolean* che indica, se impostata a *true*, la presenza davanti al testo della voce di menu di un *checkmark* (termine spesso tradotto con una locuzione che prende a prestito un termine del gergo commerciale, ovvero "segno di spunta").
- **Enabled** - proprietà di tipo *boolean* che, se impostata a *false*, rende *inattiva* una voce di menu, seppure ancora visibile e generalmente impostata da Windows ad un colore poco contrastato, di solito sul tono del grigio.
- **Visible** - proprietà di tipo *boolean* che, se impostata a *false*, rende la voce *invisibile* e, di conseguenza, *inattiva*.

Associato ad ogni oggetto di tale classe vi è un unico evento accessibile dal *property editor*, ovvero *OnClick*. La funzione di risposta a tale evento viene richiamata ogniqualvolta l'utente seleziona la voce di menu relativa.

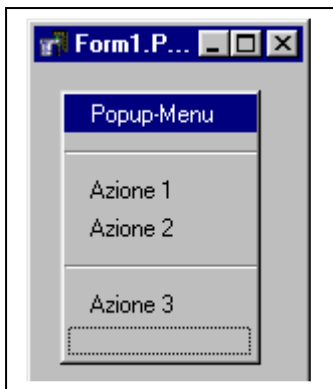
Menu e status-bar.

Nelle applicazioni professionali spesso si utilizza una status-bar per fornire utili informazioni fondamentali di aiuto all'utente.

La costruzione di una status-bar passa attraverso il componente *TPanel*, un particolare oggetto in grado di occupare un'area rettangolare, eventualmente evidenziata da bordo ed effetti tridimensionali di rilievo o incavo e collocata nella parte inferiore di una Form.



Menu pop-up.

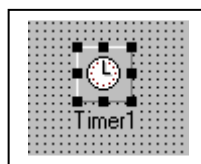


La selezione di voci di menu può risultare in alcuni casi scomoda, ad esempio quando si lavora su parti della finestra lontane dalla barra di menu. E' utile allora poter legare al tasto destro del mouse l'attivazione di un menu locale, intendendo con tale parola sia una vicinanza geometrica della posizione del menu rispetto a quella del puntatore del mouse, sia un comportamento personalizzato delle opzioni visualizzate a seconda dell'oggetto

su cui si clicca il tasto del mouse. Tali menu vengono detti *pop-up* e la classe VCL che ne incapsula proprietà, metodi ed eventi è la *TPopupMenu*.

I menu pop-up hanno molto in comune ai menu standard della classe *TMainMenu*: condividono lo stesso *menu designer*, contengono entrambi una lista di *TMenuItem* e condividono la logica di attivazione delle funzioni associate alle singole voci. La grossa differenza è che, per rendere disponibile un menu pop-up, è necessario associarlo ad un qualche controllo, attraverso la proprietà *PopupMenu*.

Eventi periodici: la classe *Timer*

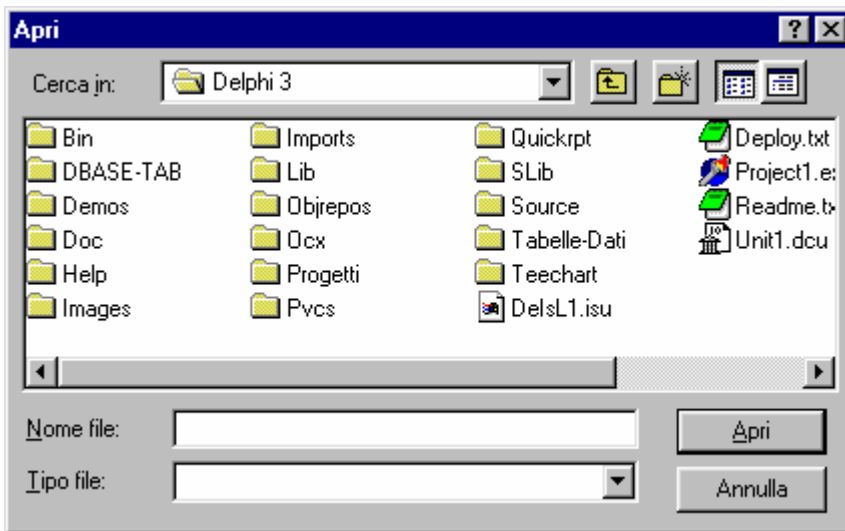


Gli oggetti *Timer* causano un evento *OnTimer* con cadenza periodica. La durata del periodo è specificata nella proprietà *interval*, espressa in millisecondi. All'interno della funzione associata a tale evento è quindi possibile compiere operazioni ripetute ad intervalli di tempo regolari.

La proprietà chiave *Enabled*, di tipo booleano, permette di abilitare o disabilitare temporaneamente il lancio degli eventi *OnTimer*. La si può sfruttare, ad esempio, per realizzare un timer di tipo *one-shot*, ovvero che scatta una sola volta e non con cadenza periodica.

Common Dialog

Delphi offre supporto alla maggior parte delle caselle comuni di dialogo fornite dai sistemi operativi con interfaccia grafica di Microsoft. Nella VCL esistono classi che rappresentano le caselle di apertura e salvataggio di file, di selezione dei font e della stampante di default, di gestione delle operazioni di ricerca e sostituzione all'interno di testo.



Oggetto OpenFileDialog (Apertura File)

Componenti Delphi dedicati all'accesso ai Database

Una delle peculiarità di Delphi è la presenza di un nutrito insieme di Oggetti di accesso ai database. Gran parte degli Oggetti della VCL, infatti, sfruttando il *Borland Database Engine (BDE)* fornisce al programmatore un facile mezzo di accesso a basi di dati sia locali, in formato *Paradox* o *dBase*, che remote, gestite da server SQL. BDE è lo strato di *middleware*, in pratica un insieme di *API* contenute all'interno di una serie di DLL, che permette di accedere in modo uniforme a basi di dati, indipendentemente sia dalla loro rappresentazione, sia dalla loro dislocazione.

Una volta configurato correttamente il motore di database, il programmatore non si deve curare minimamente dei dettagli di connessione alle tabelle che compongono l'insieme dei dati da utilizzare: le procedure di accesso, infatti, non variano.

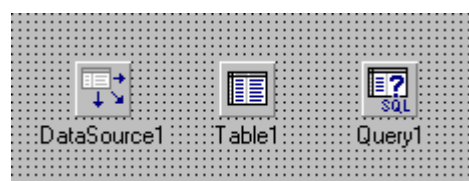
Un paragone efficace, sicuramente ben conosciuto da tutti, è quello del meccanismo di driver per stampanti oppure per schede video utilizzato in Windows: gli stessi programmi, con le chiamate alle stesse API della GDI, funzionano senza cambiamenti sia sulle schede video VGA che sull'ultima generazione di schede acceleratrici.

Ovviamente i risultati in termini di prestazioni sono enormemente differenti: quello che è importante, però, è l'aver raggiunto un ottimo grado di indipendenza dall'hardware, o comunque dalle particolari rappresentazioni interne, astruendo i concetti comuni a tutta una serie di dispositivi.

Così è per i database, grazie ad invenzioni come ODBC di Microsoft, e BDE di Borland: l'importante è rendere i propri programmi virtualmente in grado di girare su qualsiasi base di dati per cui sia disponibile un driver. E' ovvio che attraverso i driver si ha accesso ad una serie di funzionalità comuni e di base, ma comunque sufficientemente ampie per i comuni compiti di sviluppo. Nel momento in cui si necessita di velocità o di funzioni peculiari, BDE non chiude la porta e permette, ad esempio, di sfruttare peculiarità di alcuni server SQL. Ovviamente, qualora si vada a sfruttare peculiarità, si perde qualcosa in termini di portabilità: passando ad altro server, sarà necessario effettuare cambiamenti alle parti di software che implicano lo sfruttamento delle caratteristiche legate al particolare gestore di data-base.

I componenti dedicati all'accesso ai dati.

Centrale è l'oggetto *TDataSet*, che altro non è che l'astrazione di un insieme ordinato di righe di dati (record), suddivise ognuna secondo un insieme ordinato di colonne, ognuna delle quali identifica un campo all'interno del record. Generalmente non si utilizza direttamente tale classe, ma si ricorre ad uno dei discendenti *TTable*, *TQuery* oppure *TStoredProc*. Questi componenti discendono tutti dalla classe *TDBDataSet*, la quale discende a sua volta da *TDataSet*.



Come già si può intuire dalle denominazioni, *TTable* rappresenta una tabella di database, sia dal punto di vista della sua struttura (metainformazioni sulla tabella, ovvero informazioni sulle informazioni contenute nel database), sia dei record veri e propri in essa memorizzati. Normalmente, si tratta dell'astrazione relativa ad un file che contiene record fisicamente immagazzinati su di un disco. *TQuery*, invece, è rappresentazione dell'insieme di dati restituiti da una *query SQL*, ovvero una ricerca su una o più tabelle di un database espressa utilizzando un linguaggio sviluppato ad hoc per tali compiti (*Structured Query Language*). Il risultato di una query SQL, generalmente, non ha controparte diretta su disco, nel senso che i record risultanti possono essere pensati solo come insieme

temporaneo, tanto da poterlo immaginare conservato nella sola memoria centrale del computer.

All'interno delle tabelle è possibile definire una serie di *indici*, grazie ai quali è possibile mantenere ordinate, secondo il contenuto di uno o più campi, le righe delle tabelle. Non esiste alcuna classe VCL in grado di rappresentare indici: in generale le funzioni per lo sfruttamento ed il mantenimento di tali strutture di ordinamento vengono fornite direttamente dalla classe *TTable*. Essendo per natura tabelle temporanee, le query SQL non possono fornire indici per la consultazione. Vedremo comunque come sia possibile richiedere ordinamenti sull'insieme di record di risposta in sede di definizione della query stessa.

TTable.

La classe *TTable* incapsula le funzionalità proprie delle tabelle di database. Ogni volta che si ha bisogno di accedere ad una specifica tabella o ad un insieme di tabelle di database si fa riferimento ad una o più oggetti di questa classe. Per configurare correttamente le proprietà di un oggetto *TTable*, dopo averlo posizionato sulla form, occorre impostare il contenuto di alcune proprietà fondamentali, elencate di seguito.

Per accedere ad un database, nel nostro caso locale e quindi corrispondente ad una directory che contiene un insieme di file in formato dBase oppure Paradox, si deve per prima cosa impostare la proprietà *DatabaseName* immettendo o la directory in questione, oppure un *alias* definito nel Borland Database Engine. Non è necessario, una volta configurato BDE, ricordarsi i nomi degli alias disponibili: facendo click alla destra del campo che permette l'immissione della proprietà, Delphi provvede a riempire una combo-box con i nomi di tutti gli alias disponibili.

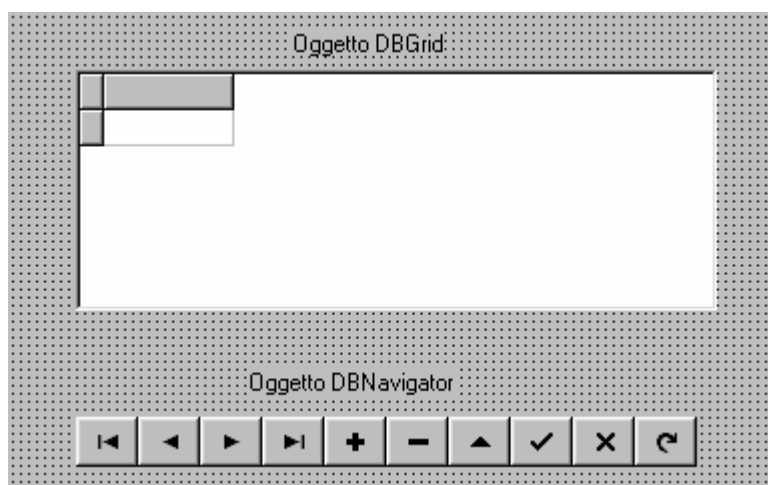
La seconda proprietà da impostare è *TableName*, ovvero il nome della tabella cui si vuole avere accesso tramite l'oggetto *TTable*. Nell'esempio, tale proprietà è stata impostata a *libri1.db*. Come per *DatabaseName* è sufficiente cliccare alla destra della proprietà per avere una lista delle tabelle disponibili all'interno del database selezionato.

Queste sono le uniche proprietà che si devono impostare obbligatoriamente per avere accesso ad una tabella: per le altre sono normalmente sufficienti le impostazioni di default. Altre proprietà utili sono la *ReadOnly* che, se impostata a valore logico vero (*true*), fa in modo che qualsiasi tentativo di cambiare i dati contenuti nella tabella fallisca: non si potrà quindi né variare il contenuto di alcun campo, né effettuare cancellazioni o inserimenti di record. La proprietà *Exclusive*, impostata a *true*, fa guadagnare accesso esclusivo alla tabella, assicurandoci quindi che nessun altro utente, ma anche nessun altro oggetto

TTable, o comunque un *Dataset*, all'interno della propria applicazione, possa accedere alla tabella. Questa proprietà si applica solo a tabelle locali, quindi non nel caso di accesso a tabelle gestite da server SQL.

Componenti "Data-bound"

I componenti più utilizzati per l'accesso a campi di database: *TDBGrid* - griglia in grado di gestire visualizzazione, editing, inserimenti e cancellazione di record all'interno di dataset; *TDBEdit* - campo di edit su di una singola colonna di tabella di database e *TDBNavigator* - insieme di speedbutton in grado di comandare la navigazione e le operazioni di editing, inserimento e cancellazione su dataset.



Nel caso si volessero solamente visualizzare i dati contenuti all'interno di uno o più record, non offrendo quindi alcuna capacità di editing all'utente, si potrebbe ricorrere alla disabilitazione dei controlli *TDBGrid* e *TDBEdit*, impostando la proprietà *Enabled* al valore logico *false*.

Per gestire campi memo, ovvero campi di testo dalla lunghezza non definita a priori dallo schema di database, è possibile fare riferimento alla classe *TDBMemo*, in grado di fornire accesso a dati suddivisi anche su più linee grazie alla proprietà *Items* di tipo *TStrings*. Essendo l'analogo data-aware di *TMemo*, è quasi superfluo affermare che *TDBMemo* condivide un largo insieme di funzioni di base con tale classe. Una proprietà interessante è la *AutoDisplay*: se impostata a valore logico vero, ogniqualvolta il record corrente cambia, viene visualizzato anche il contenuto del campo memo; in caso contrario, l'aggiornamento della visualizzazione avviene solo se si effettua un doppio click sul campo. In quest'ultimo caso, dato che, per definizione, nei campi memo vi è solitamente una grande quantità di informazione, si può godere di sostanziali benefici in termini di incremento di velocità nelle operazioni di browse interattive sul database.

Bibliografia Essenziale

Filippo Bosi – Corso delphi 2 – Edizioni Infomedia

Borland - Delphi 2 – Guida Utente