

# STANDARD DI CODIFICA DELPHI



**Project00**

Ing. M.Planchestainer

<http://members.tripod.com/mplank>

4.PJST - Standard di codifica Delphi  
Ver.1.00  
Febbraio 1999

Questo documento rappresenta lo standard di codifica Delphi di  
**Project00** realizzato sulla base dello standard Essedi [1].

Viene distribuito in forma gratuita, a condizione che venga citato il  
seguente riferimento:

M.Planchestainer – *Standard di codifica Delphi ver.1.00* -  
<http://members.tripod.com/mplank>, 1999

## COME UTILIZZARE QUESTO DOCUMENTO

Questo documento descrive un possibile standard di codifica da utilizzare per progetti Delphi.

Un qualsiasi progetto software non banale necessita di regole ben precise che governino il lavoro di sviluppo. Stabilire uno standard di codifica permette di generare moduli che si interfacciano meglio, programmi più leggibili, prodotti più facilmente mantenibili. Lo standard di codifica può essere adottato a livello di un solo progetto, di un gruppo di progetti, di un cliente oppure genericamente di tutti gli elaborati di una software house.

Questo standard è aggiornato alla versione 4 di Delphi, tuttavia il carattere generico della maggior parte delle informazioni riportate permette di utilizzarlo anche per release precedenti dell'ambiente di sviluppo.

Deve essere modificato o personalizzato per le esigenze specifiche di ogni progetto o team di sviluppo.

Questo documento si basa sullo standard di codifica del progetto ERP-Fashion di Essedi [1], il quale a sua volta si è richiamato a Inprise [4], come si evince dalla VCL, e soprattutto a Pacheco, Teixeira [2].

# SOMMARIO

<b>COME UTILIZZARE QUESTO DOCUMENTO.....</b>	<b>2</b>
<b>SOMMARIO .....</b>	<b>3</b>
<b>REGOLE GENERALI DI FORMATTAZIONE DEL CODICE SORGENTE .....</b>	<b>6</b>
INDENTAZIONE .....	6
COMMENTI .....	6
MARGINI .....	7
COPPIE BEGIN..END .....	7
<b>OBJECT PASCAL - GENERALITÀ .....</b>	<b>8</b>
PARENTESI .....	8
PAROLE RISERVATE E PAROLE CHIAVE .....	8
<b>PROCEDURE E FUNZIONI .....</b>	<b>9</b>
NOMENCLATURA/FORMATTAZIONE .....	9
FORMATTAZIONE PARAMETRI .....	9
NOMENCLATURA PARAMETRI .....	9
ORDINAMENTO DEI PARAMETRI .....	10
PARAMETRI COSTANTI .....	10
COLLISIONE DI NOMI .....	11
<b>VARIABILI .....</b>	<b>12</b>
NOMENCLATURA E FORMATTAZIONE DEI NOMI .....	12
VARIABILI LOCALI .....	12
USO DI VARIABILI GLOBALI .....	12
<b>TIPi .....</b>	<b>13</b>
CONVENZIONI DI CAPITALIZZAZIONE .....	13
TIPi A VIRGOLA MOBILE .....	13
TIPi ENUMERATI .....	13
VARIANT E OLEVARIANT .....	13
<b>TIPi STRUTTURATI .....</b>	<b>15</b>
TIPi ARRAY .....	15
TIPi RECORD .....	15
<b>ISTRUZIONI .....</b>	<b>16</b>
ISTRUZIONE IF .....	16
ISTRUZIONE CASE .....	16
ISTRUZIONE WHILE .....	16
ISTRUZIONE FOR .....	17
ISTRUZIONI REPEAT .....	17
ISTRUZIONI WITH .....	17
<b>GESTIONE STRUTTURATA DELLE ECCEZIONI .....</b>	<b>18</b>
REGOLE GENERALI .....	18
USO DI TRY. . .FINALLY .....	18
USO DI TRY. . .EXCEPT .....	18
USO DI TRY...EXCEPT...ELSE .....	19
<b>CLASSI .....</b>	<b>20</b>
NOMENCLATURA/FORMATTAZIONE .....	20
CAMPI OD ATTRIBUTI .....	20

METODI .....	20
USO DEI METODI STATICI.....	20
USO DI METODI VIRTUALI/DINAMICI .....	20
USO DI METODI ASTRATTI .....	21
METODI DI ACCESSO ALLE PROPRIETÀ .....	21
PROPRIETÀ.....	21
<b>NOMENCLATURA DI FILE .....</b>	<b>22</b>
FILE DI PROGETTO .....	22
FILE DI UNIT.....	22
CLAUSOLE USES.....	22
SEZIONE INTERFACE .....	22
SEZIONE IMPLEMENTATION .....	22
SEZIONE INIZIALIZATION.....	23
SEZIONE FINALIZATION.....	23
NOMI DEI FILE DELLE FORM .....	23
NOMI DEI FILE DI DATA MODULE.....	23
NOMI DEI FILE DI REMOTE DATAMODULE .....	23
NOMI DEI FILE DI REPORT .....	23
<b>UNIT DI USO GENERALE.....</b>	<b>24</b>
UNIT DEI COMPONENTI.....	24
INTESTAZIONE DELLE UNIT .....	24
INTESTAZIONI DI PROCEDURE/FUNZIONI .....	24
<b>FORM E DATA MODULE.....</b>	<b>26</b>
NOMENCLATURA TIPO DELLE FORM .....	26
NOMENCLATURA ISTANZA DELLE FORM.....	26
NOMENCLATURA TIPO DATAMODULE.....	26
NOMENCLATURA ISTANZA DATAMODULE .....	26
FORM/DATAMODULE AUTOCREATI .....	26
<b>PACKAGES .....</b>	<b>27</b>
USO DI PACKAGE RUNTIME O DESIGN.....	27
STANDARD DI NOMENCLATURA DEI FILE .....	27
<b>COMPONENTI.....</b>	<b>28</b>
COMPONENTI DEFINITI DALL'UTENTE .....	28
<i>Standard di nomenclatura dei tipi</i> .....	28
<i>Unit dei componenti</i> .....	28
<i>Uso delle unit di registrazione</i> .....	28
CONVENZIONI DI NOMENCLATURA DELLE ISTANZE DEI COMPONENTI .....	28
<b>PREFISSI DEI COMPONENTI .....</b>	<b>29</b>
TAB STANDARD .....	29
TAB ADDITIONAL .....	29
TAB WIN32 .....	29
TAB SYSTEM .....	30
TAB INTERNET.....	30
TAB DATA ACCESS .....	31
TAB DATA CONTROLS .....	31
TAB DECISION CUBE .....	31
TAB QREPORT .....	32
TAB QRDESIGN.....	32
TAB QRDTOOLS .....	33
TAB DIALOGS .....	33
TAB WIN31 .....	33
TAB SAMPLES .....	34
TAB ACTIVEX.....	34

TAB MIDAS .....	34
TAB RX CONTROLS.....	34
TAB RX DBAWARE .....	35
TAB RX TOOLS .....	35
TAB ADDINS.....	36
<b>RIFERIMENTI.....</b>	<b>37</b>
TABELLA DELLE REVISIONI.....	37
RIFERIMENTI BIBLIOGRAFICI.....	37

# REGOLE GENERALI DI FORMATTAZIONE DEL CODICE SORGENTE

## INDENTAZIONE

L'indentazione sarà di due spazi per livello. Non salvare i caratteri tabulazione con i file sorgente. Il motivo di questo è che i caratteri di tabulazione sono espansi con diverse larghezze per diverse impostazioni utente e per diverse utilità di gestione dei sorgenti (stampa, archiviazione, controllo versione, ecc.).

E' possibile disabilitare il salvataggio dei caratteri di tabulazione disabilitando i check box "Use Tab Character" e "Optimal fill" nella pagina Editor della finestra di dialogo "Environment options" (selezionabile tramite la voce di menu Tools|Environment).

## COMMENTI

Non usare i commenti per dire quello che già dice il codice ad un programmatore Delphi. Usare i commenti per delineare gli scopi generali del codice e per spiegare i motivi di alcune scelte fatte che potrebbero non essere più chiari a distanza di un certo periodo di tempo.

In linea di massima si dovrebbe utilizzare un metodo di scrittura del codice che parta dal PDL (Program Design Language) per poi evolvere in codice sorgente (si raccomanda l'approccio di "Code Complete" [3]).

Inserire un commento ad ogni inizio di unit e ad ogni inizio di procedura, utilizzare in special modo la parte `interface` delle unit per i commenti che poi genereranno la documentazione dei programmi, dunque per commenti riguardanti scopi, utilizzi, note particolari. La sezione `implementation` dovrebbe contenere solo commenti strettamente legati al codice, destinati a chi farà manutenzione.

Ci saranno perciò tre livelli di commento. Il primo livello sarà costituito dai commenti che devono diventare parte integrante della documentazione tecnica, da estrarre automaticamente con un tool di documentazione del tipo di DelphiDoc. Il secondo livello è costituito da commenti veri e propri del codice, per essi utilizzare le parentesi graffe. Al terzo livello si trovano i brevi commenti strettamente tecnici, per essi usare le doppie barrette.

A livello due e tre si aggiungano, in linea di massima, le iniziali o lo pseudonimo del programmatore e la data del commento.

Per disabilitare porzioni di codice utilizzare sempre le parentesi graffe con a seguire la prima parentesi i caratteri `--`, si veda esempio (ma a questo proposito è preferibile la compilazione condizionata, ove possibile).

```
{** Commento che è parte integrante
    della documentazione tecnica
}

{ commento normale - mp 10/02/1999 }

// commento strettamente tecnico - mp
```

```
{--
  Showmessage('test');
}
```

## MARGINI

---

I margini sono posti a 80 caratteri. In generale, il codice non dovrebbe eccedere questo margine con la eccezione di finire una parola, ma questa regola è abbastanza flessibile. Dove possibile, le istruzioni che superano il margine dovrebbero essere mandate a capo dopo una virgola od un operatore. Quando una istruzione è divisa su più linee, dovrebbe essere indentata di due caratteri dalla linea di codice di partenza.

## COPPIE BEGIN..END

---

L'istruzione `begin` dovrebbe apparire su una sua propria riga. Nell'esempio seguente, la prima istruzione è errata; la seconda è corretta:

```
for i := 0 to maxIter do begin      // scorretto

for i := 0 to maxIter do           // corretto, begin separato
begin
```

Una eccezione a questa regola si ha quando l'istruzione `begin` fa parte di una clausola `else`, ad esempio:

```
if qualche istruzione = then
begin
. . .
end
else begin
. . .
end;
```

L'istruzione `end` appare sempre da sola.

Quando l'istruzione `begin` non fa parte di una clausola `else`, l'istruzione `end` è sempre indentata in modo da allinearsi con il `begin` correlato.

# OBJECT PASCAL - GENERALITÀ

## PARENTESI

Non dovrebbero esserci mai spazi bianchi tra una parentesi aperta e il carattere successivo. Analogamente, non dovrebbero mai esserci spazi bianchi tra una parentesi chiusa e il precedente carattere. L'esempio seguente illustra un modo scorretto ed uno corretto d'utilizzo delle parentesi:

```
ChiamaProc( UnParametro );           // scorretto
ChiamaProc(UnParametro);             // corretto
```

Non includere mai parentesi non necessarie in una istruzione. Le parentesi dovrebbero essere usate quando richiesto per raggiungere uno scopo preciso nel codice sorgente. L'esempio seguente illustra un uso scorretto ed uno corretto:

```
if (I = minValore) then                // scorretto
if (I = minValore) or (J = maxValore) then // corretto
```

## PAROLE RISERVATE E PAROLE CHIAVE

Le parole riservate dell'Object Pascal dovrebbero essere sempre completamente minuscole.



# PROCEDURE E FUNZIONI

## NOMENCLATURA/FORMATTAZIONE

I nomi delle routine dovrebbero sempre iniziare con una lettera maiuscola e seguire la classica regola di "nessuno spazio, prima lettera di ogni parola maiuscola" per questioni di leggibilità. Il seguente è un esempio di nome di procedura formattato scorrettamente:

```
procedure questoeunnomediproceduramalformattato;
```

Questo invece è un esempi di un nome di procedura formattato correttamente:

```
procedure QuestoNomeDiProceduraEPiuLeggibile;
```

Alle routine dovrebbero essere sempre dati dei nomi esplicativi delle funzioni che svolgono e dovrebbero avere nomi in italiano, per poterle meglio distinguere da chiamate a funzioni e procedure di Delphi. Le routine che eseguono una azione dovrebbero avere come prima parola del nome il verbo dell'azione, ad esempio:

```
procedure FormattaHardDisk;
```

La lunghezza raccomandata per un nome di routine è tra 9 e 25 caratteri, non usare nomi eccessivamente abbreviati perché rendono difficile la successiva manutenzione del codice.

Come eccezione alla regola dell'italiano le routine utilizzate per impostare valori di proprietà dovrebbero, in conformità alle regole generali usate in tutto il linguaggio, essere prefissate dalla parola *Set*, per esempio:

```
procedure SetNomeUtente;
```

Allo stesso modo, le routine usate per leggere valori di proprietà dovrebbero essere sempre prefissate da *Get*, per esempio:

```
function GetNomeUtente: string;
```

## FORMATTAZIONE PARAMETRI

Dove possibile, i parametri formali dello stesso tipo dovrebbero essere combinati in una istruzione:

```
procedure TrovaPippo(Param1, Param2: Integer; Param3: String);
```

## NOMENCLATURA PARAMETRI

Tutti i nomi di parametri formali dovrebbero ricordare il loro scopo e tipicamente saranno basati sul nome dell'identificatore che è stato passato alla routine. Quando opportuno, i nomi di parametri saranno prefissati con "Un", ad esempio:

```
procedure StampaPippo(UnNomeUtente: string; UnaEtaUtente
:Integer);
```

Il prefisso "Un" è una convenzione per non fare confusione quando il nome del parametro è lo stesso di una proprietà o nome di campo nella classe.

## ORDINAMENTO DEI PARAMETRI

---

Il seguente ordinamento dei parametri formali consente una ottimizzazione del codice compilato.

I parametri più frequentemente usati (dal chiamante) dovrebbero essere ai primi posti. I parametri meno usati dovrebbero seguire i primi in un ordine da sinistra a destra.

Le liste di input dovrebbero essere prima delle liste di output in un ordine da sinistra a destra.

Piazzare i parametri generici prima dei parametri più specifici, in un ordine da sinistra a destra. Per esempio:

```
CreaCapitalePianeta(UnPianeta, UnContinente, UnaNazione,
UnaRegione, UnaCitta);
```

Piazzare i parametri senza default prima, a seguire i parametri con un valore di default, in un ordine da sinistra a destra. Solitamente i parametri che forniscono dei valori di default sono quelli di uso meno frequente, quindi questa regola non dovrebbe essere di difficile attuazione. Usare questa funzionalità soprattutto quando si stanno aggiungendo nuovi parametri ad una routine, in modo da non costringere a modificare tutte le chiamate alla procedura, anche dove il nuovo parametro non interessa.

Eccezioni alla regola di ordinamento sono possibili, come nel caso dei gestori di evento, dove un parametro chiamato Sender di tipo TObject è spesso passato come primo parametro.

## PARAMETRI COSTANTI

---

Quando i parametri di tipo record, array, ShortString o interfaccia non sono modificati da una routine, i parametri di quella routine dovrebbero essere dichiarati come `Const`. Questo consente al compilatore di generare un codice che passa i parametri non modificati nel modo più efficiente possibile.

Parametri di altro tipo possono essere opzionalmente marcati come `Const` se essi non sono modificati da una routine. Sebbene questo non abbia effetti sull'efficienza del codice, fornisce maggiori informazioni sull'uso dei parametri al chiamante della routine.

## COLLISIONE DI NOMI

---

Quando si usano due unit che contengono entrambe una routine con lo stesso nome, la routine che risiede nella unit che appare per ultima nella clausola uses verrà invocata se chiamate la routine.

Per evitare questa ambiguità, prefissare sempre tali chiamate di metodi con il loro nome di unit, come nell'esempio:

```
SysUtils.FindClose(SR);
```

oppure

```
Windows.FindClose(Handle);
```

# VARIABILI

---

## NOMENCLATURA E FORMATTAZIONE DEI NOMI

---

Le variabili dovrebbero avere dei nomi mnemonici.

Alle variabili di controllo dei cicli si dovrebbero generalmente dare dei nomi come `I`, `J`, o `K`. E' anche accettabile usare un nome più significativo, come `IndiceUtente`.

I nomi delle variabili booleane dovrebbero essere abbastanza descrittivi così che il loro assumere `True` o `False` sia significativo.

## VARIABILI LOCALI

---

Le variabili locali usate all'interno delle procedure seguono le stesse convenzioni di uso e nomenclatura di tutte le altre variabili. Alle variabili temporanee verrà dato un nome di conseguenza.

Quando necessaria, l'inizializzazione delle variabili locali verrà fatta immediatamente dopo l'entrata della routine. Le stringhe `Ansi` locali sono inizializzate automaticamente come stringhe vuote, le interfacce sono inizializzate automaticamente a `nil`, e le variabili di tipo `Variant` e `OleVariant` sono inizializzate a `Unassigned`.

## USO DI VARIABILI GLOBALI

---

L'uso di variabili globali è scoraggiato. Comunque, esse possono essere usate se necessario. Quando questo avviene, sarebbe meglio mantenere tali variabili entro il contesto in cui sono usate. Per esempio, una variabile globale potrebbe essere globale solo entro lo *scope* di una singola sezione `implementation` di una `unit`.

Dati globali usati da più di una `unit` dovrebbero essere posti in una `unit` comune usata da tutti.

I dati globali potrebbero essere inizializzati con un valore direttamente nella sezione `var`. Ricordare che tutti i dati globali sono automaticamente inizializzati a zero, così non si inizializzino le variabili globali con valori "vuoti", come `0`, `nil`, `''`, `unassigned` e così via. Un motivo di questo è che i dati globali inizializzati con zero non incrementano del dimensioni del file `.EXE`. I dati inizializzati con zero sono posti in un segmento di dati virtuale che è allocato in memoria solo al lancio dell'applicazione. Viceversa, le variabili inizializzate a codice con valori (zero o non) ingrandiscono il file `.EXE` e, di conseguenza, aumentano l'occupazione di disco.

# TIPI

## CONVENZIONI DI CAPITALIZZAZIONE

I nomi di tipo che sono parole riservate del linguaggio dovrebbero essere completamente minuscoli. I tipi `API Win32` sono generalmente tutti maiuscoli, e si dovrebbero seguire le convenzioni usate in `Windows.pas` o altre `unit` di API. Per altri nomi di variabili, la prima lettera dovrebbe essere maiuscola, così come le altre lettere inizio di nuove parole, per chiarezza. Esempi:

```
var
  MiaStringa: string;           // parola riservata
  WindowHandle: HWND;          // tipo API Win32
  I: integer;                   // identificatore (System.pas)
```

## TIPI A VIRGOLA MOBILE

L'uso del tipo `Real` è scoraggiato perché esiste solo per compatibilità con vecchio codice Pascal. Usare `double` per bisogni generici di tipi a virgola mobile. Inoltre, il tipo `double` è ottimizzato dal processore ed è un tipo di formato standard IEEE. Usare `extended` solo quando è necessario un range maggiore di quello offerto da `double`. `Extended` è un tipo specifico Intel e non è supportato da Java. Usare `single` solo quando la dimensione fisica in byte della variabile è significativa (nel caso ad esempio di DLL scritte in altri linguaggi).

## TIPI ENUMERATI

I nomi dei tipi enumerati dovrebbero essere significativi del loro scopo e contenuto. Il nome di tipo dovrebbe essere prefissato dal carattere `T` per evidenziare la sua caratteristica di tipo. Ogni componente della lista dovrebbe avere il nome prefissato da due o tre lettere minuscole che ne identificano l'appartenenza, ad esempio:

```
TTipoCanzone = (tcRock, tcClassica, tcCountry, tcNewAge,
tcHeavyMetal, tcJazz);
```

Istanze di un tipo enumerato dovrebbero avere lo stesso nome del tipo senza il prefisso `T` (`TipoCanzone`), a meno che non ci sia un particolare motivo per dare un nome più specifico, come `CanzoniPreferite1`, `CanzoniPreferite2`, e così via.

## VARIANT E OLEVariant

L'uso generalizzato di questi tipi è scoraggiato, ma il loro uso risulta necessario quando il tipo di dato è noto solo a runtime, come nel caso della programmazione COM o di database. Usare `OleVariant` per programmazione COM come Automazione o controlli ActiveX, e usare `Variant` per programmazione non-COM. La ragione di questo è che il tipo `Variant` può contenere in modo efficiente le stringhe native Delphi, mentre `OleVariant` converte tutte le stringhe in stringhe OLE (stringhe

`WideChar`) che non seguono il meccanismo del conteggio dei riferimenti, ma vengono sempre copiate.

# TIPI STRUTTURATI

## TIPI ARRAY

I tipi array dovrebbero avere dei nomi significativi. Il nome di tipo deve essere prefissato da un carattere T. Se viene dichiarato un tipo puntatore ad array, deve essere prefissato dal carattere P e deve essere dichiarato subito prima dell'array, per esempio:

```
type
  PArrayCiclico = ^TArrayCiclico
  TArrayCiclico = array[1..maxElem] of Integer;
```

Alle istanze di array si darà lo stesso nome del tipo senza il prefisso T, a meno che non si debbano dichiarare più array dello stesso tipo.

## TIPI RECORD

Ai tipi record dovrebbe essere dato un nome significativo. Il nome di tipo deve essere preceduto dal carattere T. Se si dichiara un puntatore al record, deve essere prefissato dal carattere P al posto della T e deve essere dichiarato subito prima del tipo. La dichiarazione del tipo può essere opzionalmente indentata a destra, ad esempio:

```
Type
  PImpiegato = ^TImpiegato
  TImpiegato = record
    NomeImpiegato: string
    StipendioImpiegato: Double;
  end;
```

# ISTRUZIONI

---

## ISTRUZIONE `IF`

---

La clausola di più probabile esecuzione in una istruzione `if/then/else` dovrebbe essere posta nella clausola `then`, e la clausola meno probabile nella parte `else`.

Cercare di evitare quanto possibile `if` concatenate ed usare al loro posto quando possibile istruzioni `case`.

Non indentare comunque ad una profondità maggiore di cinque livelli. Se si verifica il caso, modificare l'approccio alla soluzione.

Non usare parentesi estranee in una istruzione `if`.

Se si devono verificare condizioni multiple in una istruzione `if`, le condizioni dovrebbero essere sistemate da sinistra a destra, ordinate dalla meno alla più complicata come computazione. Questo per far sì che il codice si avvantaggi della logica short-circuit evaluation del compilatore. Per esempio, se la `Condizione1` è più veloce della `Condizione2` che è più veloce della `Condizione3`, la istruzione `if` dovrebbe essere composta così:

```
if Condizione1 and Condizione2 and Condizione3 then
```

## ISTRUZIONE `CASE`

---

Le singole possibilità all'interno di una istruzione `case` dovrebbero essere ordinate numericamente od alfabeticamente.

Le istruzioni dopo ogni singolo caso dovrebbero essere mantenute semplici e generalmente non eccedere le quattro o cinque righe di codice. Se l'azione è più complessa, il codice dovrebbe essere posto in una propria procedura o funzione.

L'uso della clausola `else` in una istruzione `case` dovrebbe essere usato solo per porre dei default o per individuare degli errori.

Le istruzioni `case` seguono le stesse regole di formattazione di altri costrutti riguardo ad indentazione e convenzioni di nomenclatura.

## ISTRUZIONE `WHILE`

---

L'uso della procedura `Exit` per uscire da un ciclo è scoraggiato; quando possibile, si dovrebbe sempre uscire dal ciclo solo con l'apposita condizione.

Tutto il codice di inizializzazione per un ciclo `while` dovrebbe essere fatto subito prima di entrare nel ciclo e non dovrebbe essere separato da altre istruzioni non correlate.

Ogni operazione di pulizia dovrebbe essere fatta subito dopo l'uscita dal ciclo.



## **ISTRUZIONE** FOR

---

Le istruzioni `for` dovrebbero essere usate al posto dei `while` quando il codice deve essere eseguito per un numero di incrementi noto.

## **ISTRUZIONI** REPEAT

---

Le istruzioni `repeat` sono simili alle istruzioni `while` e dovrebbero seguirne le stesse regole.

## **ISTRUZIONI** WITH

---

Le istruzioni `with` dovrebbero essere usate sporadicamente e con considerevole cautela. Evitare un eccessivo uso ed evitare di usare oggetti multipli, `record` e così via nelle istruzioni `with`. Per esempio:

```
with Record1, Record2 do
```

codice simile può confondere il programmatore e portare facilmente a errori difficili da rilevare.

Le istruzioni `with` seguono le stesse regole di formattazione già viste nel documento presente riguardo alle convenzioni di nomenclatura ed indentazione.

# GESTIONE STRUTTURATA DELLE ECCEZIONI

## REGOLE GENERALI

La gestione delle eccezioni dovrebbe essere usata abbondantemente sia per la correzione di errori sia per la protezione delle risorse. Questo significa che in tutti i casi in cui vengono allocate delle risorse, deve essere usato un blocco `try...finally` per assicurare un corretto rilascio delle risorse. L'eccezione a ciò si ha quando le risorse sono allocate/liberate nella parte di `initialization/finalization` di una `unit` o nei `constructor/destructor` di un oggetto.

## USO DI TRY. . .FINALLY

Dove possibile, ogni allocazione dovrebbe essere posta entro un costrutto `try. . .finally`. Per esempio, il seguente codice, passibile di bug:

```
ClasseGenerical := TClasseGenerica.Create;  
ClasseGenerica2 := TClasseGenerica.Create;  
try  
  { fa qualche cosa }  
finally  
  ClasseGenerical.Free;  
  ClasseGenerica2.Free;  
end;
```

dovrebbe essere riscritto in un modo più sicuro:

```
ClasseGenerical := TClasseGenerica.Create;  
try  
  ClasseGenerica2 := TClasseGenerica.Create;  
  try  
    { fa qualche cosa }  
  finally  
    ClasseGenerical.Free;  
  end;  
finally  
  ClasseGenerica2.Free;  
end;
```

## USO DI TRY. . .EXCEPT

Usare `try...except` solo quando si vuole svolgere qualche compito quando un'eccezione è sollevata.

In generale, non si dovrebbe usare `try. . .except` semplicemente per mostrare un codice sullo schermo poiché questo verrà fatto automaticamente nel contesto dell'applicazione dall'oggetto `Application`.

Se si vuole invocare il gestore delle eccezioni di default dopo che si è svolto qualche compito nella clausola `except`, usare `raise` per risollevare l'eccezione per il successivo gestore.

## **USO DI TRY...EXCEPT...ELSE**

---

Evitare l'uso della `else` dopo `try`. `except` perché blocca tutte le eccezioni, anche quelle per cui non si era pronti.

# CLASSI

---

## NOMENCLATURA/FORMATTAZIONE

---

I nomi di tipo per le classi dovrebbero rispecchiare lo scopo della classe. Il nome di tipo dovrebbe essere prefissato dal carattere T per evidenziarlo come definizione di tipo, ad esempio:

```
type
    TCliente = class(TObject)
```

I nomi di istanza per le classi dovrebbero generalmente essere uguali al nome del tipo senza il prefisso T, ad esempio:

```
var
    Cliente: TCliente;
```

Notare che, pur non essendo obbligatorio specificare TObject come classe padre di TCliente, sia stato comunque specificato per maggiore completezza e chiarezza del codice.

## CAMPI OD ATTRIBUTI

---

I nomi dei campi di classe seguono le stesse convenzioni degli identificatori di variabile tranne che sono prefissati dal carattere F per indicare che sono nomi di campo.

Tutti i campi dovrebbero essere privati. Campi che devono essere resi accessibili al di fuori dello *scope* della classe devono essere esportati tramite l'uso di proprietà.

## METODI

---

Le regole di nomenclatura dei metodi sono le stesse già viste nel presente documento per le procedure e le funzioni.

## USO DEI METODI STATICI

---

Usare metodi statici quando non si vuole che un metodo sia sovrascritto dalle classi discendenti

## USO DI METODI VIRTUALI/DINAMICI

---

Usare metodi virtuali quando si può voler modificare il metodo nelle classi discendenti. I metodi dinamici dovrebbero essere usati solo nelle classi per le quali ci saranno molti discendenti (diretti o indiretti). Per esempio, una classe contenente un metodo riscritto infrequentemente e 100 classi discendenti dovrebbe essere reso dinamico per ridurre l'uso di memoria per le 100 classi discendenti.

## USO DI METODI ASTRATTI

---

I metodi astratti si usano per classi che non saranno istanziate.

## METODI DI ACCESSO ALLE PROPRIETÀ

---

Tutti i metodi di accesso alle proprietà devono apparire nelle sezioni `private` o `protected` della definizione di classe.

Le convenzioni di denominazione seguono le stesse regole delle procedure e funzioni. Il metodo di accesso in lettura deve essere prefissato con la parola `Get`. Il metodo di accesso in scrittura deve essere prefissato con la parola `Set`. Il parametro per il metodo di scrittura deve avere il nome `Value` ed il suo tipo deve essere lo stesso del metodo che rappresenta, ad esempio:

```
TQualcheClasse = class(TObject)
private
    FQualcheCampo: Integer;
protected
    function GetQualcheCampo: Integer;
    procedure SetQualcheCampo(Value: Integer);
public
    property QualcheCampo: Integer read GetQualcheCampo write
        SetQualcheCampo;
end;
```

## PROPRIETÀ

---

Le proprietà che servono per esporre campi privati si chiameranno come i campi che rappresentano senza la F iniziale.

I nomi di proprietà dovrebbero essere oggetti, non verbi. Le proprietà rappresentano dati; i metodi rappresentano azioni.

Le proprietà `array` dovrebbero essere plurali; le proprietà normali dovrebbero essere singolari.

Sebbene non richiesto, è incoraggiato almeno l'uso dei metodi di accesso in scrittura per le proprietà che rappresentano un campo privato.

# NOMENCLATURA DI FILE

---

## FILE DI PROGETTO

---

Ai file di progetto verrà dato un nome descrittivo. Per esempio, un progetto per catalogare bug di programma potrebbe chiamarsi `CatalogoBug.dpr`. In ambiente multitier, i file di progetto degli application server avranno il nome che termina con "Server", mentre al contrario i file di progetto dei programmi client avranno il nome che termina con "Client".

## FILE DI UNIT

---

Per quel che riguarda i file di unit, il loro nome sarà composto essenzialmente da tre parti. Si ipotizza un'architettura modulare dell'applicazione, dove un certo numero di unit farà parte di un cosiddetto modulo.

In base a tali considerazioni, la prima parte del nome di una unit ovvero i tre o quattro caratteri iniziali indicheranno l'appartenenza della Unit ad un modulo ben preciso.

Di seguito verrà il nome esteso dell'oggetto o della famiglia di oggetti contenuti nella unit, oppure delle funzionalità offerte dalla unit.

Per finire ci sarà un eventuale suffisso del tipo "frm" per le form, "dm" per i datamodule, "rdm" per i remote data module. Per un elenco dei suffissi attualmente previsti vedere l'appendice 1 alla fine del documento.

## CLAUSOLE USES

---

La clausola `uses` nella sezione `Interface` conterrà solo le unit richieste dal codice nella sezione di interfaccia. Rimuovere ogni nome di unit estraneo che potrebbe essere stato inserito automaticamente da Delphi. La clausola `uses` nella sezione `Implementation` conterrà solo le unit richieste dal codice nella sezione di implementazione. Rimuovere tutti i nomi di unit non necessari.

## SEZIONE INTERFACE

---

La sezione `Interface` conterrà solo le dichiarazioni di quei tipi, variabili, dichiarazioni `forward` di funzioni/procedure, ecc. che devono essere accessibili da parte di unit esterne. Tutte le altre dichiarazioni dovranno andare nella sezione `Implementation`.

## SEZIONE IMPLEMENTATION

---

La sezione `Implementation` conterrà solo tutte le dichiarazioni dei tipi, variabili, procedure/funzioni, ecc. che sono di esclusivo uso da parte della unit che le contiene.

## **SEZIONE INIZIALIZATION**

---

Non piazzare codice molto pesante nella sezione di inizializzazione di una unit. Questo rallenterebbe l'apparizione della prima form.

## **SEZIONE FINALIZATION**

---

Assicurarsi di aver rilasciato qualsiasi oggetto che sia stato allocato nella sezione di `Inizialization`.

## **NOMI DEI FILE DELLE FORM**

---

I file delle form avranno lo stesso nome del file della unit corrispondente ma avranno estensione `.dfm`.

## **NOMI DEI FILE DI DATA MODULE**

---

I file dei data module avranno lo stesso nome del file della unit corrispondente ma avranno estensione `.dfm`.

## **NOMI DEI FILE DI REMOTE DATAMODULE**

---

I file dei remote data module avranno lo stesso nome del file della unit corrispondente ma avranno estensione `.dfm`.

## **NOMI DEI FILE DI REPORT**

---

Da definire

## UNIT DI USO GENERALE

---

Le unit di uso generale seguiranno le stesse regole di nomenclatura già definite prima, con l'eccezione di eventuali unit di codice recuperate da terze parti e non ridenominabili agevolmente.

### UNIT DEI COMPONENTI

---

Le unit dei componenti verranno poste in directory separate per distinguerle dalle unit di progetto. Non devono mai essere poste nella stessa directory. Alle unit dei componenti viene dato lo stesso nome del componente in esse definito. Per gli standard relativi alle nomenclature dei componenti definiti dall'utente, vedere paragrafo relativo più avanti.

### INTESTAZIONE DELLE UNIT

---

L'uso di un commento di intestazione della unit è incoraggiato per tutti i file sorgenti del progetto e dei componenti. Conterrà un messaggio di copyright, a cosa serve la unit, particolari informazioni di cui potrebbe necessitare il programmatore che dovesse usare la unit, eventuali modifiche apportate nel tempo con data e nome di chi ha fatto tali modifiche. Riguardo ai commenti, si raccomanda di non segnalare cosa il codice fa, specie se questo è banale, ma il perché certi problemi sono stati risolti in un modo piuttosto che in un altro e tutte le altre informazioni che si pensa potrebbero servire a chi, a distanza di tempo, cercasse di capire non solo cosa fa la unit ma perché lo fa in un certo modo piuttosto che in un altro. Esempio di intestazione di unit:

```
{** Unit Pippo
© Copyright 1998 Autore
Data e versione
```

```
Questa unit serve a fare questo e quello.
Fa questo in questo modo perché nell'altro
modo non funzionava.
Per usare questa unit si raccomanda prima
di dire una preghiera.
```

```
10-11-1998 Rob: l'ho modificata perché non
funzionava niente.
}
```

### INTESTAZIONI DI PROCEDURE/FUNZIONI

---

L'uso di un commento di intestazione di procedure/funzioni è raccomandato. Conterrà una breve indicazione delle operazioni svolte dal codice e indicazioni per chi dovesse utilizzarlo. Riguardo al contenuto di questi commenti valgono le stesse considerazioni già espresse per le intestazioni di unit.

Esempio di intestazione di procedura:

```
{** Procedura QuestoEQuello
© Copyright 1998 Autore
```

**ProjectOO** – ID4 - 22/02/99  
Ing. M.Planchestainer



Data e versione

Questa procedura serve a fare  
questo e quello.  
Per richiamarla bisogna mettersi  
in ginocchio sui ceci.  
}

## FORM E DATA MODULE

---

### NOMENCLATURA TIPO DELLE FORM

---

Il nome del tipo della form rispecchia in qualche modo il nome della unit che la contiene, rovesciato: i 3 o 4 caratteri che erano suffisso diventano prefisso, mentre il resto del nome rimane invariato; per cui se abbiamo una unit che si chiama BasSingoloFmr, il tipo della form sarà TfmrBasSingolo (dove Bas indica un particolare modulo dell'applicazione). Questo consente di poter avere in Delphi le unit ordinate per modulo premendo F12 (oppure dal menu: View|Units...) e le form ordinate per tipo premendo Shift+F12 (oppure dal menu: View|Forms...).

### NOMENCLATURA ISTANZA DELLE FORM

---

Le istanze delle form assumeranno lo stesso nome del loro tipo corrispondente, perdendo la T iniziale; per cui l'istanza di form dell'esempio precedente si chiamerà FmrBasSingolo.

### NOMENCLATURA TIPO DATAMODULE

---

Il nome del tipo del datamodule rispecchia in qualche modo il nome della unit che le contiene, rovesciato: i 3 o 4 caratteri che erano suffisso diventano prefisso, mentre il resto del nome resta invariato; per cui se abbiamo una unit che si chiama BasScritturaDms, il tipo della form sarà TDmsBasScrittura. Questo per gli stessi motivi già espressi nel paragrafo relativo alla nomenclatura del tipo delle form.

### NOMENCLATURA ISTANZA DATAMODULE

---

Anche per le istanze di datamodule valgono le stesse considerazioni fatte per le istanze di form, a cui si rimanda.

### FORM/DATAMODULE AUTOCREATI

---

L'unica form auto creata sarà la main form a meno che non ci sia un buon motivo per fare diversamente. Tutte le altre dovranno essere rimosse dalla lista auto-create nel dialog box Project Options. Per quel che riguarda le form modali, verrà rimossa anche la dichiarazione di variabile presente nella unit della stessa form. La parte di codice che dovrà visualizzare la form creerà una variabile locale della stessa, ne farà la visualizzazione ed al termine provvederà a distruggere la form e ad impostare a nil la variabile.

# PACKAGES

---

## USO DI PACKAGE RUNTIME O DESIGN

---

I package di runtime conterranno solo le unit e i componenti richiesti da altri componenti in quel package. Le altre unit che contengono editor di proprietà o editor di componenti o altro codice utile per il design devono essere posti in un package di design. Le unit di registrazione verranno poste in un package di design.

## STANDARD DI NOMENCLATURA DEI FILE

---

I package verranno denominati secondo il seguente schema:

**PREFISSO**Lib**vv**.pkg – design package  
**PREFISSO**Std**vv**.pkg – runtime package

Dove il *PREFISSO* si utilizza liberamente, Lib o Std identificano il tipo di package e **vv** indica la versione di Delphi per cui il package è compilato. Notare che il nome del package contiene Lib o Std per indicare se è un package di runtime o di design.

# COMPONENTI

---

## COMPONENTI DEFINITI DALL'UTENTE

---

### Standard di nomenclatura dei tipi

I componenti dovrebbero essere battezzati in modo simile alle classi così come definito nella sezione apposita del presente documento, con l'eccezione di un PREFISSO che è il prefisso standard per i componenti da sviluppati internamente ovvero personalizzati.

### Unit dei componenti

Ogni unit dei componenti dovrebbe contenere soltanto un componente principale. Vengono considerati come componenti principali tutti quelli che compaiono nella palette dei componenti. Ogni componente/oggetto ausiliario può risiedere nella stessa unit del componente principale.

### Uso delle unit di registrazione

La procedura di registrazione dei componenti dovrebbe essere rimossa dalla unit del componente ed essere posta in una unit separata. Questa unit di registrazione dovrebbe essere usata per registrare tutti i componenti, editor di proprietà, editor dei componenti, expert, ecc.

La registrazione dei componenti dovrebbe essere fatta solo nei package di design: quindi la unit di registrazione dovrebbe essere contenuta nel package di design e non in quello di runtime.

Un nome consigliato per la unit di registrazione è:

PREFISSOxxxReg.pas

Dove PREFISSO è di libero utilizzo e xxx deve essere usato per identificare il componente che viene registrato.

## CONVENZIONI DI NOMENCLATURA DELLE ISTANZE DEI COMPONENTI

---

I seguenti prefissi dovrebbero essere assegnati ai componenti standard che sono presenti in Delphi 4; sono stati inoltre aggiunti altri popolari componenti di terze parti.

## PREFISSI DEI COMPONENTI

### TAB STANDARD

Prefisso	Componente
mm	TMainMenu
pm	TPopupMenu
mmi	TMainMenuItem
pmi	TPopupMenuItem
lbl	TLabel
edt	TEdit
mem	TMemo
btn	TButton
cb	TCheckBox
rb	TRadioButton
lb	TListBox
cb	TComboBox
scb	TScrollBar
gb	TGroupBox
rg	TRadioGroup
pnl	TPanel
al	TActionList
ac	TAction

### TAB ADDITIONAL

Prefisso	Componente
bbtn	TBitBtn
sb	TSpeedButton
me	TMaskEdit
sg	TStringGrid
dg	TDrawGrid
img	TImage
shp	TShape
bvl	TBevel
sbx	TScrollBar
clb	TCheckListBox
spl	TSplitter
stx	TStaticText
ctb	TControlBar
cht	TChart

### TAB WIN32

Prefisso	Componente
tbc	TTabControl
pgc	TPageControl

il	TImageList
re	TRichEdit
tbr	TTrackBar
prb	TProgressBar
ud	TUpDown
hk	THotKey
ani	TAnimate
dtp	TDateTimePicker
mcl	TMonthCalendar
tv	TTreeView
lv	TListView
hdr	THeaderControl
stb	TStatusBar
tlb	TToolBar
clb	TCoolBar
psc	TPageScroller

## TAB SYSTEM

---

Prefisso	Componente
tm	TTimer
pb	TPaintBox
mp	TMediaPlayer
olec	TOleContainer
ddcc	TDDEClientConv
ddci	TDDEClientItem
ddsc	TDDEServerConv
ddsi	TDDEServerItem

## TAB INTERNET

---

Prefisso	Componente
csk	TClientSocket
ssk	TServerSocket
wbd	TWebDispatcher
pp	TPageProducer
tp	TQueryTabProducer
dstp	TDataSetTableProducer
nmdt	TNMDayTime
nec	TNMEcho
nf	TNMFinger
nftp	TNMFtp
nhttp	TNMHttp
nMsg	TNMMsg
nmsg	TNMMSGServ
nntp	TNMNntp
npop	TNMPop3
nuup	TNMUUProcessor

smtp	TNMSMTP
nst	TNMStrm
nsts	TNMStrmServ
ntm	TNMTime
nudp	TNMUdp
psk	TPowerSock
ngs	TNMGeneralServer
html	THtml
url	TNMUrl
sml	TSimpleMail

## TAB DATA ACCESS

---

Prefisso	Componente
ds	TDataSource
tbl	TTable
qry	TQuery
sp	TStoredProc
db	TDataBase
ssn	TSession
bm	TBatchMove
usql	TUpdateSQL
nstt	TNestedTable

## TAB DATA CONTROLS

---

Prefisso	Componente
dbg	TDbGrid
dbn	TDbNavigator
dbt	TDbText
dbe	TDbEdit
dbm	TDbMemo
dbi	TDbImage
dblb	TDbListBox
dbcb	TDbComboBox
dbch	TDbCheckBox
dbrg	TDbRadioGroup
dbll	TDbLookupListBox
dblc	TDbLookupComboBox
dbre	TDbRichEdit
dbcg	TDbCtrlGrid
dbch	TDbChart

## TAB DECISION CUBE

---

Prefisso	Componente
dcb	TDecisionCube

dcq	TDecisionQuery
dcs	TDecisionSource
dcp	TDecisionPivot
dcg	TDecisionGrid
dcgr	TDecisionGraph

## TAB QREPORT

---

Prefisso	Componente
qr	TQuickRep
qrsd	TQRSubDetail
qrsb	TQRStringBand
qrb	TQRBand
qrcb	TQRChildBand
qrg	TQRGroup
qrl	TQRLabel
qrt	TQRText
qre	TQRExpr
qrs	TQRSysData
qrm	TQRMemo
qrem	TQRExprMemo
qrrt	TQRRichText
qrdr	TQRDbRichText
qrsh	TQRShape
qri	TQRImage
qrdb	TQRDbImage
qrcr	TQRCompositeReport
qrp	TQRPreview
qrch	TQRChart
qrtf	TQRTextFilter
qrcf	TQRCSVFilter
qrhf	TQRHTMLFilter

## TAB QRDESIGN

---

Prefisso	Componente
qrdd	TReportDesignerDialog
qrdp	TReportPrinterDialog
qrdl	TQRDLoader
qrd	TQRepDesigner
qrddq	TDesignQuickReport
qrdsd	TQRDesignSubdetail
qrdb	TQRDesignBand
qrdcb	TQRDesignChildBand
qrdg	TQRDesignGroup
qrdl	TQRDesignLabel
qrdb	TQRDesignDbText
qrde	TQRDesignExpr



qrds	TQRDesignSysData
qrdm	TQRDesignMemo
qrdsh	TQRDesignShape
qrdim	TQRDesignImage
qrddi	TQRDesignDbImage
qrdrtr	TQRDesignRichText
qrddr	TQRDesignDbRichText

## TAB QRDTOOLS

---

Prefisso	Componente
qrdeh	TQRExprHandler
qrdes	TEventScrollbox
qrdrp	TRulerPanel
qrdli	TLanguageInifile
qrdel	TEnhancedListbox

## TAB DIALOGS

---

I componenti dialog box sono in realtà delle form incapsulate in un componente. Per questo motivo, essi seguiranno delle convenzioni di nomenclatura simili a quelle delle form. La definizione del tipo è già definita per il nome del componente. Il nome di istanza sarà lo stesso del nome di tipo senza il prefisso numerico, che è assegnato da Delphi. Seguono esempi:

Tipo	Nome istanza
TOpenDialog	OpenDialog
TSaveDialog	SaveDialog
TOpenPictureDialog	OpenPictureDialog
TSavePictureDialog	SavePictureDialog
TFontDialog	FontDialog
TColorDialog	ColorDialog
TPrintDialog	PrintDialog
TPrintSetupDialog	PrintSetupDialog
TFindDialog	FindDialog
TReplaceDialog	ReplaceDialog

## TAB WIN31

---

Prefisso	Componente
dbll	TDBLookupList
dblc	TDBLookupCombo
ts	TTabSet
ol	TOutline
tnb	TTabbedNotebook
nb	TNoteBook
hdr	THeader

flb	TFileListBox
dlb	TDirectoryListBox
dcb	TDriveComboBox
fcg	TFilterComboBox

## TAB SAMPLES

---

Prefisso	Componente
gg	TGauge
cg	TColorGrid
spb	TSpinButton
spe	TSpinEdit
dol	TDirectoryOutline
cal	TCalendar
ibea	TIBEventAlerter

## TAB ACTIVEX

---

Prefisso	Componente
cfx	TChartFX
vsp	TVSSpell
f1b	TF1Book
vtc	TVTChart
grp	TGraph

## TAB MIDAS

---

Prefisso	Componente
prv	TProvider
cds	TClientDataSet
dcom	TDCOMConnection
corb	TCORBAConnection
olee	TOLEEnterpriseConnection
sck	TSocketConnection
dprv	TDatasetProvider
sob	TSimpleObjectBroker
rms	TRemoteServer
mid	TMidasConnection

## TAB RX CONTROLS

---

Prefisso	Componente
rxce	TRxComboEdit
rxfe	TRxFilenameEdit
rxde	TRxDirectoryEdit
rxdt	TRxDateEdit
rxca	TRxCalcEdit

rxcu	TRxCurrencyEdit
rxtl	TRxTextListBox
rxcl	TRxCheckListBox
rxfc	TRxFontComboBox
rxcc	TRxColorComboBox
rxsp	TRxSplitter
rxsl	TRxSlider
rxlb	TRxLabel
rxck	TRxClock
rxai	TRxAnimatedImage
rxdg	TRxDrawGrid
rxsb	TRxSpeedButton
rxga	TRxGIFAnimator
rxsn	TRxSpinButton
rxse	TRxSpinEdit
rxsw	TRxSwitch
rxdi	TRxDice

## TAB RX DBAWARE

---

Prefisso	Componente
rxqy	TRxQuery
rxss	TRxSQLScript
rxmt	TRxMemoryTable
rxqbe	TRxQBEQuery
rxdf	TRxDBFilter
rxdbg	TRxDBGrid
rxll	TRxDBLookupList
rxlc	TRxDBLookupCombo
rxle	TRxDBLookupEdit
rxdd	TRxDBDateEdit
rxdc	TRxDBCalcEdit
rxdce	TRxDBComboEdit
rxdsl	TRxDBStatusLabel
rxdcB	TRxDBComboBox
rxdr	TRxDBRichEdit
rxdic	TRxDBIndexCombo
rxdbp	TRxDBProgress
rxdse	TRxDBSecurity
rxbdi	TRxBDEItems
rxdbi	TRxDBDatabaseItems
rxtdi	TRxTableItems

## TAB RX TOOLS

---

Prefisso	Componente
rxpc	TPicClip
rxfs	TFormStorage

rxfp	TFormPlacement
rxwh	TRxWindowHook
rxae	TAppEvents
rxspb	TRxSpeedBar
rxclc	TRxCalculator
rxtm	TRxTimerList
rxpm	TRxPageManager
rxmru	TMRUManager
rxscp	TSecretPanel
rxsth	TStrHolder
rxtri	TRxTrayIcon
rxmm	TRxMainMenu
rxpo	TRxPopupMenu
rxfm	TRxFolderMonitor
rxcv	TClipboardViewer
rxgc	TRxGradientCaption
rxldd	TRxDualListDialog
rxcnv	TConverter

## TAB ADDINS

---

Prefisso	Componente
axl	TAdvExcel
xls	TExcel
Jon	TJustOne
Joa	TJustOneAsk
Aex	TAppExec
Ema	TEmail

## RIFERIMENTI

### TABELLA DELLE REVISIONI

Versione	Autore Principale	Descrizione della versione	Completata
0.92	MP, RI	Beta 1	09/02/1999
0.96	MP	Beta 2 – revisione del codice e degli esempi	16/02/1999
0.97	MP	Beta 3 – adeguamento standard, revisione esempi	19/02/1999
0.98	MP	Beta 4 – revisione nomenclatura file	22/02/1999
0.99	MP	Pre-release di test	22/02/1999
1.00	MP	Prima release ufficiale	22/02/1999

### RIFERIMENTI BIBLIOGRAFICI

Questo standard è stato ricavato da:

- [1] R.Icardi – *Standard di codifica ESSEDI Delphi 4, Progetto ERP-Fashion* – Essedi Sviluppo Clienti S.r.l., 1998
- [2] Pacheco, Teixeira - *Delphi 4 Developer's Guide* – SAMS, 1998
- [3] S.McConnell – *Code Complete* – Microsoft Press, 1993
- [4] Inprise – *VCL source code* – Inprise, 1998