Programmare in C su Amiga (30)

di Dario de Judicibus (MC2120)

La versione 1.3 di Intuition non mette a disposizione del programmatore un set di funzioni ad alto livello per la gestione dei controlli, ma fornisce una serie di strutture che permette di definire praticamente una varietà infinita di oggetti da usare nelle interfacce utente. In questa puntata incominceremo a vedere come si può concretizzare questa grande potenzialità in un insieme di funzioni e macro che rendano più semplice definire e gestire vari tipi di controlli

Introduzione

Nella scorsa puntata abbiamo parlato di pulsanti. Abbiamo visto come si definiscono, abbiamo descritto i tipi principali e ne abbiamo mostrato l'utilizzo. Abbiamo inoltre visto come Intuition ci dia una grande flessibilità nella definizione di questi controlli, ma nel contempo costringa il programmatore a gestire tutta una serie di parametri, la cui manutenzione può risultare pesante qualora si desideri modificare l'aspetto dell'interfaccia utente che si sta disegnando. Se ad esempio abbiamo deciso di visualizzare un pulsante con su scritto **Lista**, bisognerà fornire ad Intuition, fra le altre cose, una larghezza per il pulsante sufficiente a far entrare tale parola nell'area di selezione. Se ad un certo punto decidiamo di modificare il testo del pulsante in **Lista...**, ci toccherà modificare tale valore e, se il pulsante in questione si trova in mezzo ad altri pulsanti, ricentrarlo diversamente se non addirittura spostare di una decina di pixel gli altri. Il tutto potrebbe richiedere un paio di compilazioni fintanto che non si è raggiunto il risultato voluto. Naturalmente esistono dei programmi che ci permettono di fare questo direttamente per mezzo di una interfaccia grafica, per poi generare il codice finale, ma a questo punto non si parla più di programmazione, ma di *CASE*, cioè *Computer Aided Software Engineering*. Un altro aspetto della questione è che se si devono definire molti controlli, è necessario definire molte strutture, e questo aumenta le dimensioni del programma. Potrebbe risultare conveniente allora allocare le strutture necessarie in modo dinamico, e magari deallocarle quando non ci servono più. Se stiamo scrivendo un programmino semplice con qualche menu, un paio di quadri ed una finestra, la cosa non è poi così importante, ma se quello che stiamo scrivendo è un programma complesso la cosa è un po' differente. Un programma che gira in un sistema multitasking deve essere molto educato. Ad esempio non deve tenersi per sé più memoria di quanta gliene serva.

Abbiamo già visto, quando parlavamo di menu, come si possano allocare dinamicamente le strutture necessarie alla definizione delle voci. In quel caso usammo la funzione **AllocRemember()**, poiché in tal modo ci liberavamo della necessità di tener traccia dei singoli blocchi di memoria allocati in più riprese, e potevamo liberare tutta la memoria allocata fino a quel momento con un'unica chiamata alla **FreeRemember()**. Questa volta torneremo ad usare le funzioni più classiche **AllocMem()** e **FreeMem()**, in quanto vogliamo aver la possibilità di allocare e liberare singoli blocchi di memoria, associati ognuno ad un ben definito pulsante.

Opereremo in modo analogo a quanto già fatto con il programma scheletro. Anzi, useremo proprio tale struttura, opportunamente modificata, per costruire il nostro programma di gestione dei controlli. In pratica struttureremo il nostro programma in tre parti:

gdgmain.c

che rappresenterà il programma principale, e conterrà il codice di definizione della finestra, dei menu e dei controlli, di gestione degli eventi trasmessi da Intuition, delle eccezioni e dei messaggi (analogamente a quanto faceva **sklmain.c**).

gdgprocs.c

che conterrà le varie funzioni e macro *di servizio* per la definizione dei controlli e la loro gestione (analogamente a quelle relative ai menu e contenute in **sklprocs.c**);

gdgusrh.c

che conterrà le costanti, i tipi, le strutture ed i prototipi necessari al programma per la definizione e la gestione dei controlli, e che serve a generare la tabella simbolica **gdgusrh.sym** (analogamente a come si faceva con **sklusrh.c**).

Ovviamente continueremo a fare uso della **sklprocs.c** e della **sklusrh.sym** per la parte relativa ai menu, ma questo lo vedremo nella prossima puntata.

Una considerazione importante. Costruire un set di funzioni e di macro per la gestione dei controlli [*toolbox, toolkit*], richiede la necessità di definire una serie di standard, cioè di regole per la definizione dei vari oggetti.

Questo vuol dire porre delle limitazioni al programmatore che usa questi strumenti, ma in compenso aumenta notevolmente la flessibilità del programma e riduce i tempi di manutenzione. D'altra parte chi programma può sempre decidere di fare a meno di questi strumenti a tornare a programmare, per così dire in modo *nativo*, rinunciando però così, ai vantaggi che essi forniscono. In pratica si tratta di decidere che cosa ci interessa di più: se la flessibilità del programma, o quella degli strumenti.

Più cose uno strumento permette di fare, più informazioni il programmatore deve fornire e più sono le interrleazioni tra i vari oggetti nel programma. Questo rende il programma più difficile da mantenere in quanto anche un minimo cambiamento può richiedere la modifica di tutta una serie di elementi ad esso collegati. Viceversa, riducendo le possibilità ed utilizzando strumenti di alto livello, cioè *non atomici*, il programma risulta più flessibile nei confronti di eventuali variazioni, e quindi meno costoso da mantenere.

Chi ad esempio ha iniziato a programmare con la versione 2.0 di Intuition, difficilmente accetterà di tornare a programmare con la 1.3. Ma vediamo tutto ciò in pratica.

Funzioni e macro

In figura 1 sono riportate le varie dichiarative su cui si basano le procedure di gestione dei pulsanti. Torneremo più volte su questo file. Per il momento diamo un'occhiata ai prototipi delle funzioni. Abbiamo quattro funzioni:

CreateButton()
Serve a definire in modo dinamico un pulsante a rilascio automatico o manuale. Restituisce il puntatore al pulsante.

DeleteButton()
Rimuove il pulsante creato con la **CreateButton()** e libera tutta la memoria allocata per quel pulsante.

DisplayGadget()
Visualizza un pulsante aggiungendolo alla lista dei controlli relativi ad un certo contenitore (finestra o quadro).

RefreshWindow()
Restaura una finestra e tutti i controlli in essa contenuti.

A queste si aggiungono cinque macro, e cioè:

CreateAutoButton()
Serve a definire in modo dinamico un pulsante a rilascio automatico. Restituisce il puntatore al pulsante.

DeleteAutoButton()
Rimuove il pulsante creato con la **CreateAutoButton()** e libera tutta la memoria allocata per quel pulsante.

CreateToggleButton()
Serve a definire in modo dinamico un pulsante a rilascio manuale. Restituisce il puntatore al pulsante.

DeleteToggleButton()
Rimuove il pulsante creato con la **CreateToggleButton()** e libera tutta la memoria allocata per quel pulsante.

ToggleButtonStatus()
Cambia lo stato di selezione di un pulsante a rilascio manuale.

Vediamo quali sono i criteri alla base del disegno proposto.
Innanzi tutto tutte le funzioni e le macro descritte seguono le raccomandazioni della Commodore relativamente alla gestione dei pulsanti. Esse peraltro non fanno parte del sistema operativo, né di quello attuale, né della versione 2.0, e

```
/*****************************************************************************
**  Programmare in C su Amiga  (c) 1991  Dario de Judicibus - Roma (I)     **
** ------------------------------------------------------------------------ **
**  Definizioni da precompilare per generare la tabella GDGUSRH.SYM        **
*****************************************************************************/

/*
**  Costanti
*/
 #define TOGGLEON  1
 #define TOGGLEOFF 0
 #define AUTOBUTTONCLASS   0x0001
 #define TOGGLEBUTTONCLASS 0x0002

/*
**  Tipi
*/
 typedef struct Gadget       IGDG;
 typedef struct Border       IBRD;

 typedef struct Container
 {
   struct Window *w;
   struct Requester *r;
 }
 ICNT;

 typedef struct UsrButton
 {
   USHORT Number;
   USHORT Size;
 }
 UBUT;

/*
**  Macro di servizio
*/
 #define CreateAutoButton(id,txt,cont,x,y) \
    CreateButton((id),(txt),(cont),(x),(y),AUTOBUTTONCLASS,TOGGLEOFF)
 #define DeleteAutoButton(b,c) DeleteButton((b),(c))
 #define CreateToggleButton(id,txt,cont,x,y,status) \
    CreateButton((id),(txt),(cont),(x),(y),TOGGLEBUTTONCLASS,(status))
 #define DeleteToggleButton(b,c) DeleteButton((b),(c))
 #define ToggleButtonStatus(b)   (BOOL)((b)->Flags & SELECTED)

/*
**  Prototipi delle funzioni di servizio
*/
 IGDG *CreateButton  ( USHORT, char *, ICNT *, SHORT, SHORT, USHORT, BOOL );
 void  DeleteButton  ( IGDG *, ICNT * );
 void  DisplayGadget ( IGDG *, ICNT * );
 void  RefreshWindow ( struct Window *);
```

Figura 1 - gdgusrh.c.

```
/*****************************************************************************
**  Programmare in C su Amiga  (c) 1991  Dario de Judicibus - Roma (I)     **
** ------------------------------------------------------------------------ **
**  Scheletro di un programma di gestione dei controlli                     **
**                                                                          **
**    Questo file contiene le procedure di tipo generale utilizzate         **
**    dal programma principale. Questa suddivisione permette un             **
**    ulteriore risparmio di tempo in fase di compilazione, dato che        **
**    queste procedure vengono di solito modificate molto più               **
**    raramente di quelle del programma principale. In particolare          **
**    questo file contiene le procedure di gestione dei controlli.          **
**                                                                          **
**  Queste procedure sono autosufficienti ma vanno compilate sempre con     **
**  l'opzione -Hskl.sym, in modo da utilizzare gli headers precompilati.    **
**                                                                          **
*****************************************************************************/


/*
**  Costanti
*/
 #define EOGL ((USHORT) 0)
 #define POINTER(type,base,offset) (type *)((UBYTE *)(base)+(offset))
 #define SWAPPENS(penA,penB,temp) {temp = penA, penA = penB, penB = temp;}

/*
**  Macro
*/
 #define GDGMEM  (MEMF_CLEAR|MEMF_CHIP)
 #define ITXTL(itxt) IntuiTextLength((itxt))

/*
**  Strutture modello
*/
 ITXT textModel =      /* Questa struttura fa da base ai testi (controlli)*/
 {
   1, 0,               /* Penne per il tratto e per lo sfondo            */
   JAM2, 5,            /* Modo grafico e spostamento dal bordo sinistro  */
   0,                  /* Spostamento dal bordo superiore                */
   NULL,               /* Font usato, se diverso da quello di sistema.   */
   NULL,               /* Testo (da riempire)                            */
   NULL                /* Eventuale struttura successiva                 */
 };

 IBRD rectModel =      /* Questa struttura fa da base ai bordi           */
 {
   0, 0,               /* Posizione rispetto il contenitore              */
   1, 0, JAM1,         /* Penne (tratto e sfondo) e modo grafico         */
   5,                  /* Numero di punti per il rettangolo              */
   NULL,               /* Vettore delle coordinate                       */
   NULL                /* Eventuale struttura successiva                 */
 };
```

Figura 2 - gdgprocs.c: definizioni.

non sono descritte in alcun manuale o libro di programmazione su Intuition. Si tratta di procedure da me scritte apposta per questa rubrica e quindi seguono una impostazione personale che può o meno essere condivisa. Per questo motivo ritengo importante vedere il perché di determinate scelte, in modo che, se esse non rispecchiano quelle che sono le vostre esigenze, possiate avere una chiara indicazione di cosa e come modificare il codice qui riportato.

Per prima cosa, se un certo parametro può essere generato automaticamente dal programma sulla base dei dati forniti, od in base ad una serie di criteri standard, esso è eliminato dalla lista di quelli che il programmatore deve specificare, riducendo così consistentemente il numero di informazioni da fornire direttamente alle varie funzioni.

Secondo. Le strutture necessarie alla definizione dei vari controlli sono *tutte* allocate dinamicamente, in modo da rendere le dimensioni del programma indipendenti dal numero di controlli che esso utilizza.

Tre. Pur non essendo il disegno propriamente *object-based*, ho cercato di eliminare nella definizione dei controlli l'utilizzo di variabili globali. Se una informazione relativa ad un controllo deve poter essere utilizzata da un'altra procedura o dal programma principale, tale informazione deve essere incapsulata nel controllo stesso. Per far ciò ho utilizzato la possibilità che Intuition mette a disposizione dei programmatori di agganciare in fondo alla struttura **Gadget** una struttura utente, definibile a piacere.

Quattro. Ogni tipo di controllo deve avere un set di funzioni per crearlo e distruggerlo, ed eventualmente un set di funzioni per gestirne il comportamento, se questo non è gestito direttamente da Intuition. D'altra parte, dato che lo stesso tipo di funzione per vari tipi di controlli può richiedere più meno le stesse operazioni, queste sono raggruppate in singole funzioni relative a più controlli, salvo poi fornire delle macro che differenziano la funzione a seconda delle varie tipologie. Cinque. Le varie funzioni devono essere indipendenti dal tipo di contenitore usato per il controllo (finestra o quadro), e dal posizionamento del controllo nel contenitore (corpo o bordo della finestra). Per questo motivo si è separata la funzione che crea il controllo, da quella che lo visualizza nel corpo di una finestra.

CreateButton()

Questa funzione (vedi figura 3) serve a creare dinamicamente un pulsante a rilascio automatico o manuale. Vediamola in dettaglio.

Essa riceve come dati in ingresso sette parametri:

id
l'identificativo del pulsante;

txt
il testo da visualizzare nel pulsante;

container
il puntatore ad una struttura *contenitore* (vedi più avanti);

x, y
la posizione nel contenitore del pulsante (più avanti sono spiegati in dettaglio i valori che tali coordinate possono assumere, ed il loro significato);

class
il tipo di pulsante, se cioè a rilascio automatico o manuale;

onoff
lo stato iniziale del pulsante (significativo solo per i pulsanti a rilascio manuale).

Per l'occasione ho definito una nuova struttura, chiamata **Container**. Questa struttura, definita in **gdgusrh.c** (vedi figura 1), contiene il puntatore al contenitore del pulsante ed alla finestra a cui esso è associato, se è un quadro. Ovviamente, se il contenitore è una finestra, il puntatore al quadro è nullo.

Per prima cosa la funzione alloca un'area di memoria che poi suddivide in otto aree dati. In questo modo si evita di allocare e poi liberare ogni singola area dati. Le otto aree sono rispettivamente:

- una struttura **Gadget**
- una struttura **IntuiText**
- quattro strutture **Border**
- due vettori da cinque coordinate l'uno.

Si associa quindi, per comodità, un puntatore ad ogni area dati, utilizzando la macro **POINTER** definita nella parte dichiarativa della **gdgprocs.c**. Questa macro permette di definire un puntatore di qualunque tipo a partire da un altro puntatore, non necessariamente dello stesso tipo, e da uno spostamento in byte [*offset*] rispetto a quest'ultimo.

A questo punto si definiscono quei campi della struttura **Gadget** che non vanno calcolati, come l'identificativo del controllo ed il tipo di controllo (cioè un *pulsante*). Se poi si tratta di un pulsante a rilascio automatico, chiederemo che emetta solo eventi di tipo **GADGETUP**, altrimenti dovrà emettere solo eventi di tipo **GADGETDOWN**, come già spiegato nella scorsa puntata. In quest'ultimo caso, ovviamente, il pulsante sarà a rilascio manuale (**TOGGLESELECT**). Inoltre i pulsanti definiti qui sono visualizzati utilizzando dei bordi, e non delle immagini. La tecnica di evidenziazione sarà inoltre quella a bordi alternati. Associamo quindi al controllo i puntatori a due delle quattro strutture bordo precedentemente definite ed alla struttura **IntuiText**. In seguito ne definiremo il contenuto.

Vi ricordate quel puntatore disponibile ai programmatori per estendere la struttura **Gadget** a seconda delle esigenze del programma? Bene, assegniamo a tale puntatore una struttura chiamata **UsrButton**, definita in **gdgusrh.c** (figura 1), e che, per il momento contiene il numero di pulsanti associati a questo controllo, e le dimensioni dell'area di memoria allocata per *tutte* le strutture relative allo stesso.

Il primo valore ci servirà quando definiremo dei *gruppi*, cioè più controlli legati logicamente insieme, come i pulsanti mutualmente esclusivi. Il secondo ci permetterà di liberare tutta la memoria allocata con un'unica chiamata, come vedremo più avanti. Chiameremo questa area, *area dati privata del pulsante*. Proprio nell'ottica di definire in seguito gruppi di controlli, o controlli multipli, cioè controlli *differenti* costruiti in modo da formare un unico controllo, non useremo sempre la tecnica di definire una singola catena di controlli utilizzando il campo **NextGadget**, ma useremo tale tecnica *solo* per legare fra loro controlli appartenenti allo stesso gruppo.

A questo punto inizia la parte che definisce i restanti campi in modo automatico. Innanzi tutto se il pulsante fa parte di un quadro o se deve essere selezionato in partenza, vengono modificati opportunamente i relativi campi già definiti precedentemente. Quindi si definisce la struttura **IntuiText**. Questo viene fatto copiando nell'area dati assegnata a tale struttura, una struttura modello definita in precedenza. Quindi si assegna il testo da visualizzare nel pulsante, e se ne calcola altezza e larghezza. A questo punto si calcola il *centraggio* del testo nel pulsante, lasciando un bordo a destra e sinistra pari ad un ottavo della lunghezza del testo, e sopra e sotto uno spazio pari ad un terzo dell'altezza del testo. Ovviamente ognuno può scegliersi i valori che vuole. Per quello che mi riguarda, preferisco definirli proporzionalmente al testo stesso, in modo da ottenere un risultato esteticamente proporzionato al font usato ed alla lunghezza del testo. Da notare che l'altezza del testo è ricavata da uno dei campi della struttura **Window** in modo dinamico, mentre per la lunghezza viene usata la **IntuiTextLenght()**.

Per quello che riguarda invece la posizione del pulsante nel contenitore, ho deciso di utilizzare un criterio differente da quello utilizzato dal sistema. Uno degli aspetti più delicati nel disegno di una interfaccia è infatti quello di definire la posizione dei singoli controlli, in modo da ottenere un aspetto esteticamente accettabile, funzionalmente pratico e, se

```
/*****************************************************************************
** CreateButton()                FUNZIONE              Versione 1.00       **
**                                                                         **
** Funzione fornita:  crea dinamicamente un pulsante a rilascio            **
**                    manuale.                                             **
**                                                                         **
** Dati in ingresso:  id          identificativo del pulsante              **
**                    txt         testo del pulsante                       **
**                    container   contenitore (finestra e/o quadro)        **
**                    x, y        posizione del pulsante nel contenitore   **
**                    class       classe del pulsante (tipo rilascio)      **
**                    onoff       stato iniziale del pulsante              **
**                                                                         **
** Dati in uscita:    Gdg         puntatore al pulsante                    **
**                                                                         **
** Dati globali:      User        memorizza in Size la memoria utilizzata  **
**                                                                         **
** Eventi emessi:     GADGETDOWN  per la classe AUTOBUTTONCLASS            **
**                    GADGETUP    per la classe TOGGLEBUTTONCLASS          **
**                                                                         **
*****************************************************************************/
IGDG *CreateButton(id,txt,container,x,y,class,onoff)
  USHORT id ;
  char *txt ;
  ICNT *container ;
  SHORT x, y ;
  USHORT class ;
  BOOL onoff ;
{
 /*
 **  Allocheremo otto aree dati:
 **
 **  -- una struttura Gadget
 **  -- una struttura IntuiText
 **  -- quattro strutture Border
 **  -- due vettori da cinque coordinate
 **
 */
  IGDG *Gdg ;
  ITXT *Txt ;
  IBRD *InnBrdUns, *InnBrdSel, *OutBrdUns, *OutBrdSel ;
  UBUT *User ;
  SHORT *InnCoord, *OutCoord ;
  USHORT GdgSize, TxtSize, BrdSize, CrdSize, UsrSize, TotSize ;
  UBYTE SwapPen ;
  SHORT i, l, h ;

 /*
 **  Calcola le dimensioni delle varie aree
 */
  GdgSize = sizeof(IGDG) ;
  TxtSize = sizeof(ITXT) ;
  BrdSize = sizeof(IBRD) ;
  CrdSize = sizeof(SHORT)*10 ;
  UsrSize = sizeof(UBUT) ;
  TotSize = GdgSize + TxtSize + 4*BrdSize + 2*CrdSize + UsrSize ;

 /*
 **  Alloca memoria per il pulsante e le strutture collegate
 */
  Gdg = (IGDG *)AllocMem(TotSize, GDGMEM) ;
  if (Gdg == NULL) return(NULL) ;
  Txt       = POINTER( ITXT, Gdg,       GdgSize) ;
  InnBrdUns = POINTER( IBRD, Txt,       TxtSize) ;
  InnBrdSel = POINTER( IBRD, InnBrdUns, BrdSize) ;
  OutBrdUns = POINTER( IBRD, InnBrdSel, BrdSize) ;
  OutBrdSel = POINTER( IBRD, OutBrdUns, BrdSize) ;
  InnCoord  = POINTER(SHORT, OutBrdSel, BrdSize) ;
  OutCoord  = POINTER(SHORT, InnCoord , CrdSize) ;
  User      = POINTER( UBUT, OutCoord , CrdSize) ;

 /*
 **  Assegna i campi fissi della struttura Gadget
 */
  Gdg->NextGadget = NULL ;          /* Questo è un controllo singolo      */
  Gdg->GadgetID   = id ;            /* Identificativo del controllo       */
  Gdg->GadgetType = BOOLGADGET ;    /* Tipo di controllo                  */
  Gdg->Flags      = GADGHIMAGE ;    /* Dato che è evidenziato da un bordo */
  if (class == AUTOBUTTONCLASS)     /* Se è a rilascio automatico, allora */
    Gdg->Activation = RELVERIFY ;   /* mi interessa sapere se rilasciato  */
  if (class == TOGGLEBUTTONCLASS)   /* Se è a rilascio manuale, allora    */
    Gdg->Activation = GADGIMMEDIATE /* mi interessa sapere quando premuto */
                    | TOGGLESELECT ;/* (pulsante a rilascio manuale)      */
  Gdg->MutualExclude = NULL ;       /* Non utilizzato -- per sicurezza -- */
  Gdg->SpecialInfo   = NULL ;       /* Non utilizzato -- per sicurezza -- */
  Gdg->GadgetText   = Txt ;
  Gdg->GadgetRender = (APTR)InnBrdUns ;
  Gdg->SelectRender = (APTR)InnBrdSel ;
  Gdg->UserData     = (APTR)User ;  /* Questa è una struttura di servizio */
  User->Size        = TotSize ;     /* Qui metto quanta memoria ho preso  */
  User->Number      = 1 ;           /* Numero di pulsanti nel gruppo      */
 /*
 **  Il pulsante fa parte di un quadro ?
 */
  if (container->r) Gdg->GadgetType |= REQGADGET ;

 /*
 **  E' selezionato in partenza ?
 */
  if (onoff) Gdg->Flags |= SELECTED ;

 /*
 **  Associa al pulsante il suo testo
 */
  *Txt = textModel ;                /* Copia il prototipo del controllo */
  Txt->IText = (UBYTE *)txt ;       /* Copia il testo vero e proprio    */
  l = ITXTL(Txt) ;                  /* Lunghezza del testo              */
  h = container->w->RPort->TxHeight ; /* Altezza del testo              */
  Txt->LeftEdge = l/8 ;             /* Ascissa del testo nel controllo  */
  Txt->TopEdge  = h/3 ;             /* Ordinata del testo nel controllo */
 /*
 **  Alloca i campi variabili della struttura Gadget
 */
  Gdg->Width    = l + 2*Txt->LeftEdge ;
  Gdg->Height   = h + 2*Txt->TopEdge ;

 /*
 **  Per rendere più semplice la vita al programmatore, se un campo è
 **  negativo, lo consideriamo una coordinata rispetto al bordo del
 **  controllo PIU' VICINO a quello da cui si parte a misurare.
 */
  if (x > 0)                        /* Ascissa rispetto ad...           */
  {
    x += container->w->BorderLeft ;
    Gdg->LeftEdge = x ;                          /* ...il bordo sinistro */
  }
  else
  {
    x -= container->w->BorderRight ;
    Gdg->LeftEdge = x - Gdg->Width ;             /* ...il bordo destro   */
    Gdg->Flags |= GRELRIGHT ;
  }
  if (y > 0)                        /* Ordinata rispetto ad...          */
  {
    y += container->w->BorderTop ;
    Gdg->TopEdge  = y ;                          /* ...il bordo superiore */
  }
  else
  {
    y -= container->w->BorderBottom ;
    Gdg->TopEdge  = y - Gdg->Height ;            /* ...il bordo inferiore */
    Gdg->Flags |= GRELBOTTOM ;
  }

 /*
 **  Definisci le strutture bordo
 */
  *InnBrdUns = rectModel ;
  *InnBrdSel = rectModel ;
  *OutBrdUns = rectModel ;
  *OutBrdSel = rectModel ;

  if (class == AUTOBUTTONCLASS)     /* Pulsante a rilascio automatico     */
  {
    SWAPPENS(OutBrdUns->FrontPen,OutBrdUns->BackPen,SwapPen) ;
    SWAPPENS(InnBrdSel->FrontPen,InnBrdSel->BackPen,SwapPen) ;
  }

  if (class == TOGGLEBUTTONCLASS)   /* Pulsante a rilascio manuale        */
  {
    InnBrdUns->FrontPen = 2; /* Se il pulsante non è selezionato usa un   */
    InnBrdSel->FrontPen = 2; /* colore diverso da quello dei pulsanti a   */
    OutBrdUns->FrontPen = 0; /* rilascio automatico. Se è selezionato usa */
    OutBrdSel->FrontPen = 2; /* un doppio bordo per evidenziarlo meglio.  */
  }

  InnBrdUns->NextBorder = OutBrdUns ;
  InnBrdSel->NextBorder = OutBrdSel ;
  OutBrdUns->NextBorder = NULL ;
  OutBrdSel->NextBorder = NULL ;

  InnBrdUns->XY = InnCoord ;
  InnBrdSel->XY = InnCoord ;
  OutBrdUns->XY = OutCoord ;
  OutBrdSel->XY = OutCoord ;

 /*
 **  Definisci i vettori di coordinate
 */
  InnCoord[2] = Gdg->Width - 1 ;  InnCoord[4] = Gdg->Width - 1 ;
  InnCoord[5] = Gdg->Height - 1 ; InnCoord[7] = Gdg->Height - 1 ;

  for (i=0; i<10; i++)
    OutCoord[i] = InnCoord[i] + (InnCoord[i] ? 1 : -1) ;

 /*
 **  Fatto. Il pulsante è pronto.
 */
  return (Gdg) ;
  }
```

Figura 3 - gdgprocs.c: CreateButton().

```
/*************************************************************************
** DeleteButton()              FUNZIONE          Versione 1.00        **
**                                                                    **
** Funzione fornita:  cancella dinamicamente un pulsante a rilascio   **
**                    automatico.                                     **
**                                                                    **
** Dati in ingresso:  button      puntatore al pulsante               **
**                    container   contenitore (finestra e/o quadro)   **
**                                                                    **
** Dati in uscita:    nessuno                                         **
**                                                                    **
** Dati globali:      User        ricava da Size la memoria utilizzata **
**                                                                    **
*************************************************************************/
void DeleteButton(button,container)
  IGDG *button ;
  ICNT *container ;
{
 /*
 **  Rimuovi il pulsante dal contenitore
 */
  (void)RemoveGList(container->w,button,1) ;
  RefreshWindow(container->w) ;

 /*
 **  Cancella la memoria per il pulsante e le strutture collegate
 */
  FreeMem(button,(((UBUT *)(button->UserData))->Size)) ;
}
```

Figura 4 - gdgprocs.c: DeleteButton().

```
/*************************************************************************
** CreateAutoButton()          MACRO             Versione 1.00        **
**                                                                    **
** Funzione fornita:  crea dinamicamente un pulsante a rilascio       **
**                    automatico.                                     **
**                                                                    **
** Dati in ingresso:  id          identificativo del pulsante         **
**                    txt         testo del pulsante                  **
**                    container   contenitore (finestra e/o quadro)   **
**                    x, y        posizione del pulsante nel contenitore **
**                                                                    **
** Dati in uscita:    Gdg         puntatore al pulsante               **
**                                                                    **
** Dati globali:      User        memorizza in Size la memoria utilizzata **
**                                                                    **
** Eventi emessi:     GADGETUP                                        **
**                                                                    **
*************************************************************************
**                                                                    **
** Definita in:  gdgusrh.c                                            **
**                                                                    **
** #define CreateAutoButton(id,txt,cont,x,y) \                       **
**    CreateButton((id),(txt),(cont),(x),(y),AUTOBUTTONCLASS,TOGGLEOFF) **
**                                                                    **
*************************************************************************/
```

Figura 5 - gdgprocs.c: CreateAutoButton().

```
/*************************************************************************
** DeleteAutoButton()          MACRO             Versione 1.00        **
**                                                                    **
** Funzione fornita:  cancella dinamicamente un pulsante a rilascio   **
**                    automatico.                                     **
**                                                                    **
** Dati in ingresso:  button      puntatore al pulsante               **
**                    container   contenitore (finestra e/o quadro)   **
**                                                                    **
** Dati in uscita:    nessuno                                         **
**                                                                    **
** Dati globali:      User        ricava da Size la memoria utilizzata **
**                                                                    **
*************************************************************************
**                                                                    **
** Definita in:  gdgusrh.c                                            **
**                                                                    **
** #define DeleteAutoButton(b,c) DeleteButton((b),(c))               **
**                                                                    **
*************************************************************************/
```

Figura 6 - gdgprocs.c: DeleteAutoButton().

possibile, intuitivo. Molti hanno l'abitudine di disegnare i vari elementi dell'interfaccia su un foglio di carta millimetrata o, in mancanza di un generatore automatico di codice, sullo schermo per mezzo di un prodotto tipo *DPaint* od anche un *CAD*. Si tratta comunque di un processo elaborato, di fatto eccessivo se non si sta sviluppando un grosso progetto, e che richiede tempo e pazienza. All'altro estremo c'è la tecnica *ad occhio*, in cui si danno le coordinate degli oggetti in modo approssimato, per poi modificarle leggermente ad ogni compilazione, fintanto che non si è ottenuto il risultato voluto. Questo processo è più rapido se gli oggetti sono pochi, e sfrutta il fatto che comunque, nel processo di sviluppo di un programma, è necessario spesso ricompilare più volte un programma a causa dei vari errori che immancabilmente esso contiene e che vanno quindi eliminati. Questa seconda tecnica è sufficientemente accurata se si definisce la posizione relativamente ad un altro oggetto vicino, e se questa non dipende dalle dimensioni dell'oggetto.

Supponiamo ora di avere un pulsante con su scritto *Premini!* Se lo vogliamo posizionare in alto a sinistra nel corpo di una finestra, possiamo definire le coordinate dell'origine del pulsante (**10,10**), ad esempio, assumendo di avere una finestra abbastanza larga ed alta. Ma se lo vogliamo posizionare nell'angolo in basso a destra? Allora bisogna calcolare la lunghezza e l'altezza del pulsante, aggiungervi la distanza che lo stesso deve avere rispettivamente dal bordo destro e da quello inferiore della finestra, e sottrarre il tutto dalle dimensioni della finestra stessa, come si può vedere in figura 12. Per ovviare a ciò, la **CreateButton()** si comporta nel modo seguente:

Se le ascisse sono positive, si riferiscono alla distanza del bordo sinistro del pulsante dal bordo sinistro del contenitore, se sono negative, si riferiscono alla distanza del bordo destro del pulsante dal bordo destro del contenitore. Analogamente, se le ordinate sono positive, allora si riferiscono alla distanza del bordo superiore del pulsante dal bordo superiore del contenitore, se sono negative, si riferiscono alla distanza del bordo inferiore del pulsante dal bordo inferiore del contenitore.

In aggiunta a questo, la distanza viene calcolata già tenendo conto delle dimensioni dei bordi stessi. *A questo riguardo il codice della* **CreateButton()** *contiene un errore. Provate a capire qual è. La risposta nella prossima puntata.*

L'unico controllo che non viene effettuato automaticamente è che il pulsante, così posizionato, entri effettivamente nella finestra. Questo è importante specialmente se la finestra che lo contiene può essere ristretta dall'utente. In questo caso sta al programmatore gestire direttamente il problema.

Come già detto in precedenza, i pulsanti sono disegnati utilizzando un bordo, così come un bordo alternato è utilizzato per l'evidenziazione. Ho deciso di definire uno standard sia per il colore dei bordi, che per la forma degli stessi. Vediamo quale, premesso che il colore di fondo è «**0**» mentre il testo è reso con il colore «**1**».

Il bordo normale per i pulsanti a rilascio automatico è formato da un rettangolo tracciato con il colore «**1**» delle dimensioni dell'area di selezione, inscritto

```
/*****************************************************************************
** CreateToggleButton()          MACRO                  Versione 1.00      **
**                                                                         **
** Funzione fornita:  crea dinamicamente un pulsante a rilascio            **
**                    manuale.                                             **
**                                                                         **
** Dati in ingresso:  id          identificativo del pulsante              **
**                    txt         testo del pulsante                       **
**                    container   contenitore (finestra e/o quadro)        **
**                    x, y        posizione del pulsante nel contenitore   **
**                    onoff       stato iniziale del pulsante              **
**                                                                         **
** Dati in uscita:    Gdg         puntatore al pulsante                    **
**                                                                         **
** Dati globali:      User        memorizza in Size la memoria utilizzata  **
**                                                                         **
** Eventi emessi:     GADGETDOWN                                           **
**                                                                         **
*****************************************************************************
**                                                                         **
** Definita in:  gdgusrh.c                                                 **
**                                                                         **
** #define CreateToggleButton(id,txt,cont,x,y,status) \                   **
**    CreateButton((id),(txt),(cont),(x),(y),TOGGLEBUTTONCLASS,(status))   **
**                                                                         **
*****************************************************************************/
```

Figura 7 - gdgprocs.c: CreateToggleButton().

```
/*****************************************************************************
** DeleteToggleButton()          MACRO                  Versione 1.00      **
**                                                                         **
** Funzione fornita:  cancella dinamicamente un pulsante a rilascio        **
**                    manuale.                                             **
**                                                                         **
** Dati in ingresso:  button      puntatore al pulsante                    **
**                    container   contenitore (finestra e/o quadro)        **
**                                                                         **
** Dati in uscita:    nessuno                                              **
**                                                                         **
** Dati globali:      User        ricava da Size la memoria utilizzata     **
**                                                                         **
*****************************************************************************
**                                                                         **
** Definita in:  gdgusrh.c                                                 **
**                                                                         **
** #define DeleteToggleButton(b,c) DeleteButton((b),(c))                  **
**                                                                         **
*****************************************************************************/
```

Figura 8 - gdgprocs.c: DeleteToggleButton().

in un rettangolo di colore «**0**» appena di un pixel più grande. Quello alternato è invece formato da un rettangolo tracciato con il colore «**0**» delle dimensioni dell'area di selezione, inscritto in un rettangolo di colore «**1**» appena di un pixel più grande. La necessità di utilizzare due bordi bicolore invertiti, nasce dal fatto che, come diremo più avanti, i pulsanti sono disegnati nella stessa *bitmap* della finestra, e quindi è necessario aggiungere al rettangolo *visibile* del bordo, un rettangolo dello stesso colore dello sfondo che cancelli il rettangolo visibile del bordo alternato (e viceversa, ovviamente).

Il bordo normale per i pulsanti a rilascio manuale è formato da un rettangolo tracciato con il colore «**2**» delle dimensioni dell'area di selezione, inscritto in un rettangolo di colore «**0**» appena di un pixel più grande. Quello alternato è invece formato da due rettangoli, entrambi disegnati con il colore «**2**», uno dentro l'altro, in modo da rendere lo stesso rettangolo del bordo normale con un tratto più spesso. Il motivo di tale scelta è il seguente. Innanzi tutto così l'utente sa subito se il pulsante è a rilascio automatico o manuale, in quanto essi usano colori differenti per i bordi. Inoltre, nel caso dei pulsanti a rilascio manuale, deve essere immediato all'utente se un pulsante è nello stato selezionato o meno.

Ora, mentre per i pulsanti a rilascio automatico il risultato finale è quello di vedere il bordo del pulsante allargarsi e poi restringersi quando l'utente lo seleziona con il mouse, per quelli a rilascio manuale, due bordi dello stesso spessore che differiscono di poco nelle dimensioni non sono certo un modo di evidenziare chiaramente in quale stato si trova il pulsante. Ecco il perché del doppio bordo colorato per questo tipo di pulsanti. Provare per credere.

I bordi sono quindi quattro per ogni tipo di pulsante raggruppati a due a due. Anche in questo caso, ho utilizzato un bordo come modello da copiare prima di effettuare le variazioni descritte.

A questo punto il gioco è fatto, e la funzione restituisce il puntatore al punsante così definito.

DeleteButton()

Questa funzione (vedi figura 4) distrugge un pulsante creato con la funzione precedente. Esso rimuove prima il pulsante della lista dei controlli associati alla finestra, quindi restaura la finestra stessa (corpo e bordo) ed i controlli rimasti. A questo punto, utilizzando l'informazione contenuta nella struttura *privata* del pulsante, libera in un sol colpo *tutta* la memoria relativa al controllo, indipendentemente dal modo in cui essa era suddivisa. Questo è corretto solo se, come in questo caso, tutte le strutture contenute nell'area liberata sono referenziate solo all'interno dell'area stessa. In caso contrario è necessario prima annullare tutti i puntatori di strutture esterne che al momento della deallocazione stiano ancora puntando all'interno dell'area da rendere di nuovo disponibile al sistema.

Notate che questa tecnica permette di modificare la **CreateButton()** aggiungendo nuove strutture all'area di memoria allocata dinamicamente, od estendendo quelle attualmente definite, senza dover modificare conseguentemente la funzione **DeleteButton()**, dato che l'informazione della memoria allocata è incapsulata nell'oggetto stesso. Questo è uno dei vantaggi dell'adozione di tecniche basate sugli oggetti [*object-based*], che trova la sua massima espressione nella metodologia OO [*object-oriented*], in cui l'allocazione e la deallocazione degli oggetti è del tutto trasparente al programmatore, ed ogni oggetto racchiude in sé sia la procedura che i dati che lo riguardano.

Naturalmente qui ci stiamo ancora muovendo secondo uno stile *procedurale*, e quindi «classico». Anche così, tuttavia, possiamo utilizzare alcune tecniche di questo tipo, che ci fanno risparmiare molto tempo, sia in fase di programmazione che di manutenzione del codice.

DisplayGadget()

Questa funzione (vedi figura 10) aggiunge un controllo al contenitore e quindi lo visualizza. Come si può facilmente vedere, essa non è strettamente legata ai pulsanti, né serve solo per i controlli direttamente associati ad una finestra. Grazie all'utilizzo della più generale struttura **Container**, essa può essere utilizzata anche per aggiungere un controllo ad un quadro, purché appunto il secondo puntatore di questa struttura non sia nullo.

RefreshWindow()

Questa funzione (vedi figura 11) serve a ripristinare quella parte della finestra che possa essere stata *danneggiata* da un controllo appena rimosso. Infatti, al contrario di quello che succede in altri sistemi, come il *Presentation Manager*

```
/*************************************************************************
** ToggleButtonStatus()          MACRO                Versione 1.00      **
**                                                                       **
** Funzione fornita:  ritorna lo stato di un pulsante a rilascio manuale **
**                                                                       **
** Dati in ingresso:  button      puntatore al pulsante                  **
**                                                                       **
** Dati in uscita:    status      TOGGLEON oppure TOGGLEOFF              **
**                                                                       **
** Dati globali:      User        ricava da Size la memoria utilizzata   **
**                                                                       **
*************************************************************************
**                                                                       **
** Definita in:  gdgusrh.c                                               **
**                                                                       **
** #define ToggleButtonStatus(b) (BOOL)((b)->Flags & SELECTED)           **
**                                                                       **
*************************************************************************/
```

Figura 9 - gdgprocs.c: ToggleButtonStatus().

```
/*************************************************************************
** DisplayGadget()               FUNZIONE             Versione 1.00      **
**                                                                       **
** Funzione fornita:  aggiunge il controllo al contenitore e quindi lo   **
**                    visualizza                                         **
**                                                                       **
** Dati in ingresso:  gadget      puntatore al controllo                 **
**                    container   contenitore (finestra e/o quadro)      **
**                                                                       **
** Dati in uscita:    nessuno                                            **
**                                                                       **
*************************************************************************/
void DisplayGadget(gadget,container)
  IGDG *gadget ;
  ICNT *container ;
{

 /*
 **  Aggiungi il pulsante al contenitore
 */
  (void)AddGList(container->w,gadget,EOGL,1,container->r) ;
  RefreshGList(gadget,container->w,container->r,1) ;
}
```

Figura 10 - gdgprocs.c: DisplayGadget().

```
/*************************************************************************
** RefreshWindow()               FUNZIONE             Versione 1.00      **
**                                                                       **
** Funzione fornita:  restaura una finestra e tutti i controlli in essa  **
**                    contenuti.                                         **
**                                                                       **
** Dati in ingresso:  win         puntatore alla finestra                **
**                                                                       **
** Dati in uscita:    nessuno                                            **
**                                                                       **
*************************************************************************/
void RefreshWindow(win)
  struct Window *win ;
{
  IGDG *first ;

  SetRast(win->RPort,0) ;   /* Ridipingiamo lo sfondo della finestra      */
  RefreshWindowFrame(win) ; /* Ricostruiamo i bordi che abbiamo distrutto */
  first = win->FirstGadget ;/* Primo controllo per questa finestra        */
  if (first)                          /* Restaura tutti i controlli */
    (void)RefreshGList(first,win,NULL,EOGL) ;
}
```

Figura 11 - gdgprocs.c: RefreshWindow().

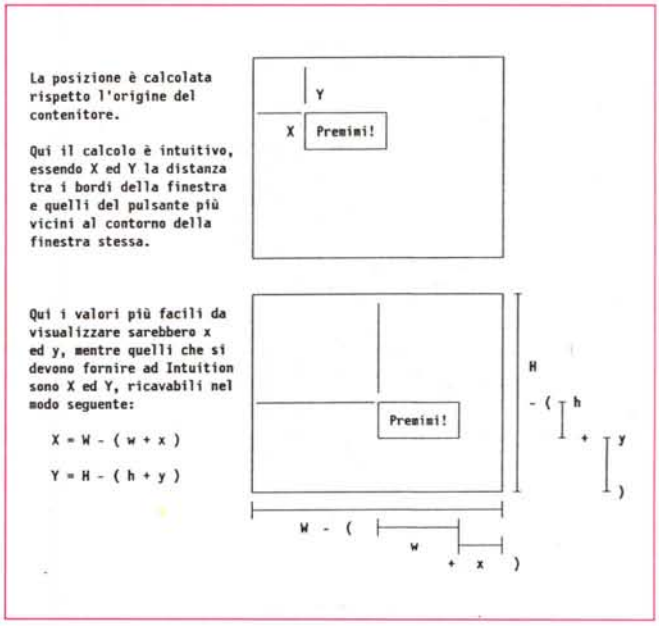

Figura 12 - Posizionamento classico di un pulsante.

dell'OS/2, i pulsanti non sono mantenuti in una *bitmap* separata, come avviene anche nell'Amiga per i menu, ma sono disegnati nella stessa *bitmap* della finestra. Quindi quando si toglie un pulsante dal corpo di una finestra, il disegno del bordo, o l'immagine utilizzata rimangono.

Ovviamente il *vero* pulsante, cioè l'area di selezione, non c'è più. È necessario allora ripristinare a mano l'area in questione, anche se si sta lavorando con una finestra a restauro automatico.

Per far questo abbiamo usato una tecnica *brutale*, che ridipinge lo sfondo della finestra, quindi ricostruisce i bordi distrutti da questa operazione e rivisualizza tutti i controlli ancora associati ad essa.

Questo in quanto, a meno di non utilizzare una finestra GZZ, anche i bordi, come gli altri controlli posizionati nel corpo della finestra, fanno parte della stessa *bitmap*.

Ovviamente voi potete scrivervi una funzione più sofisticata, oppure gestire il tutto nelle funzioni **CreateButton()** e **DeleteButton()**.

Ad esempio, se volete visualizzare un pulsante in una finestra che contiene una immagine, senza rovinare la stessa, potete estendere la struttura privata al pulsante aggiungendovi un puntatore ad un'area di memoria grande come il pulsante e con lo stesso numero di piani della finestra, trasferirvi lì l'area che deve essere ricoperta dal pulsante e, al momento di rimuovere il pulsante, ritrasferire nella finestra l'area coperta. Magari potreste provarci come esercizio.

CreateAutoButton()

Questa macro (vedi figura 5) permette di chiamare la funzione **CreateButton()** già predisposta per la creazione di un pulsante a rilascio automatico.

DeleteAutoButton()

Questa macro (vedi figura 6) permette di chiamare la funzione **DeleteButton** () già predisposta per la rimozione di un pulsante a rilascio automatico.

CreateToggleButton()

Questa macro (vedi figura 7) permette di chiamare la funzione **CreateButton()** già predisposta per la creazione di un pulsante a rilascio manuale.

DeleteToggleButton()

Questa macro (vedi figura 8) permette di chiamare la funzione **DeleteButton()** già predisposta per la rimozione di un pulsante a rilascio manuale.

ToggleButtonStatus()

Questa macro (vedi figura 9) permette di invertire lo stato di selezione di un pulsante a rilascio manuale. Va utilizzata, come vedremo nella prossima puntata, nella procedura di gestione degli eventi emessi dai pulsanti a rilascio manuale, o da quelli a rilascio automatico e che fanno parte di un gruppo di pulsanti mutualmente esclusivi.

Conclusione

Bene. Penso che adesso abbiate materiale a sufficienza per costruirvi il vostro set di funzioni per la definizione dei pulsanti.

Provate a creare nuovi tipi di pulsanti e magari provate a modificare le funzioni proposte in modo da usare immagini anziché bordi, magari seguendo lo stile del WorkBench 2.0.

Mi spiace di dover escludere da questa puntata la rubrica *Casella Postale*, anche perché ho ricevuto moltissime lettere e molti messaggi via *MC-Link*. Purtroppo lo spazio è quello che è, e penso che sarebbe stato pesante spezzettare l'argomento di questa puntata su due numeri di *MC*.

Ci rivediamo con le vostre lettere nella prossima puntata.

A presto, quindi.

MC

La scheda tecnica

Ecco i quattro comandi dell'*AmigaDos 1.3* di questo mese, a partire da SETDATE.

Nota

Nella scorsa puntata avevo promesso di parlare delle caratteristiche di rientranza e di riusabilità dei programmi. Per ragioni di spazio non mi è stato possibile includere l'argomento suddetto in questa puntata. Cercherò di metterlo nella prossima. Mi scuso con i lettori per questo cambiamento, peraltro dettato da incontestabili necessità editoriali.

LEGENDA	
<parametro>	parametro da specificare
[<opzione>]	parametro opzionale
{<opz-rip>}	parametro opzionale che può essere ripetuto n volte
...	serie che può essere continuata
|	separatore per una lista di opzioni di cui una almeno VA specificata
/A	indica che il parametro DEVE essere specificato
/K	indica che quella determinata parola chiave VA specificata se si vuole usare l'opzione ad essa associata
/S	indica una parola chiave da specificare per attivare l'operazione ad essa associata

Comando:	SETDATE
Formato:	SETDATE <file>|<indirizzario> [<data>] [<ora>]
Sintassi:	SETDATE "FILE/A,DATE,TIME"
Scopo:	Cambia la data e l'ora associata ad un file od un indirizzario
Specifiche:	Al contrario della versione precedente, non è più necessario specificare uno zero nella data, se il giorno è compreso fra 1 e 9 inclusi. Se non sono forniti nè la data nè l'ora, il comando utilizza come valori quelli correnti, cioè quelli dell'orologio del sistema [touch].
Esempio:	SETDATE fred 15-Mar-1990 12:27

Comando:	SETENV
Formato:	SETENV <variabile> <valore>
Sintassi:	SETENV "NAME/A,STRING"
Scopo:	Serve a specificare il valore di una variabile di sistema
Specifiche:	Serve ad associare ad una variabile globale un determinato valore, a cambiarne uno precedentemente specificato, od a rimuovere la variabile stessa. Nella versione 1.3 queste variabili sono memorizzate nel disco RAM: sotto l'indirizzario "env", al quale va assegnato l'identificativo ENV:. In seguito ENV: diventerà un "handler" vero e proprio. La variabile viene rimossa se non si specifica alcun valore, anche se, per ora, il nome della variabile rimane in RAM:env. Se il valore contiene spazi o caratteri speciali interpretabili da CLI o da SHELL, bisogna includere il valore fra doppie virgolette. L'utente può anche modificare la "startup-sequence" in modo da associare ENV: ad un altro indirizzario, anche su disco fisso, ma non è consigliabile per ragioni di prestazioni.
Esempio:	SETENV Dictionary SYS:WP/lex/italiano.dct

Comando:	SETPATCH
Formato:	SETPATCH [R]
Sintassi:	SETPATCH "R/S"
Scopo:	Fissa alcuni problemi nelle ROM 1.2 ed 1.3 del KickStart
Specifiche:	Deve essere lanciato come PRIMA istruzione della procedura "startup-sequence", e permette di fissare alcuni problemi esistenti nelle ROM del KickStart 1.2 ed 1.3 e che non potrebbero altrimenti essere eliminati se non sostituendo le stesse. I bachi riguardano la DisplayAlert(), i vettori delle eccezioni matematiche del 68000, DeleteLayers(), ed AllocEntry(). L'opzione R va usata in quei sistemi con 1Mb di memoria CHIP per evitare effetti collaterali sulla ramdrive.device.

Comando:	SKIP
Formato:	SKIP [<edichetta>] [BACK]
Sintassi:	SKIP "LABEL,BACK/S"
Scopo:	Salto incondizionato all'edichetta specificata.
Specifiche:	Comando da utilizzare solo nelle macro di sistema (SCRIPT). Opera un salto incondizionato all'istruzione LAB identificata dall'edichetta fornita, o da una edichetta nulla, se non ne è stata specificata alcuna. La ricerca avviene a partire dalla istruzione corrente. Se è stata specificata l'opzione BACK, la ricerca avviene dalla prima istruzione della macro, a meno che questa non contenga comandi EXECUTE. In tal caso, il salto all'indietro non va oltre all'istruzione che chiama EXECUTE.
Esempio:	SKIP pippo