TABLE OF CONTENTS

NAME
ActiveREI -- Trova la REI attiva.

SYNOPSIS
rei = ActiveREI()
D0

struct REI *ActiveREI(void);

FUNCTION
Questa funzione restituisce il puntatore alla struttura REI attiva.
Significa che restituisce la struttura REI alla quale è collegata
in quel momento la Window attiva.

Di conseguenza questo comando ammette la presenza di una Window
selezionata e attiva. Nel caso REMOTO che la REI sia composta solo
da uno schermo (caso limite) questo comando FALLISCE.

INPUTS

RESULT
rei - Indirizzo di una struttura REI. Se NULL significa che la Window
correntemente attiva non è stata aperta dal sistema REI, quindi
ERRORE.

EXAMPLE

NOTES

BUGS

SEE ALSO

NAME
AddBitPlanes -- Addiziona un bitplane ad una BitMap.

SYNOPSIS
```
num = AddBitPlanes (bm, num)
D0                  A0  D0

ULONG AddBitPlanes (struct BitMap *, UWORD);
```

FUNCTION
Addiziona uno o più bitplanes ad una struttura BitMap. Questa deve già essere inizializzata, in quanto deve contenere tutti i dati per l'addizione dei bitplanes. I bitplanes allocati, vengono puliti e posto in CHIP RAM.

(V41.1) - Ora AddBitPlanes() controlla che il numero di planes non superi quello di 8. Otto è il massimo di planes che può essere addizionato, se quindi una BitMap ha 6 bitplanes e si addizionano 3 planes, NESSUN planes verrà addizionato alla BitMap, e la funzione ritorna NULL, con un errore quindi.

INPUTS
bitmap - Indirizzo di una struttuta BitMap inizializzata.
num - Numero di bitplanes da addizionare. La BitMap non può avere più di otto (8) planes, questo valore deve quindi essere scelto con cautela, facendo riferimento ai planes attualmente presenti nella BitMap.

RESULT
num - Se NULL nessun bitplane è stato addizionato, altrimenti ritorna il numero dei bitplanes che si è potuto allocare. Di solito questo valore corrisponde a num inserito negli INPUTS. In caso differisca, significa che non c'è stata abbastanza CHIP RAM, per allocare i bitplanes richiesti, oppure il numero di bitplanes da addizionare era troppo elevato rispetto a quelli supportati dalla BitMap.

EXAMPLE

NOTES
Una facile strada per essere sicuri del numero di planes che si sta allocando, è quella di guardare nel campo bm_Depth della struttura BitMap. Questa operazione può essere fatta in due modi. O leggendo il campo bm_Depth direttamente nella struttura BitMap, o utilizzare (come consigliato dalla Commodore) graphics.library/GetBitMapAttr().

I bitplanes allocati, vengono inseriti nella struttura BitMap, a partire dalla posizione ritenuta vuota. Ciò viene fatto in base ai bitplanes presenti nella struttura BitMap.

BUGS
Questa funzione per ora non utilizza graphics.library/GetBitMapAttr() sarà aggiornata in future versioni della libreria.

SEE ALSO
RemoveBitPlanes()
graphics.library/AllocBitMap(), AllocRaster(), GetBitMapAttr()

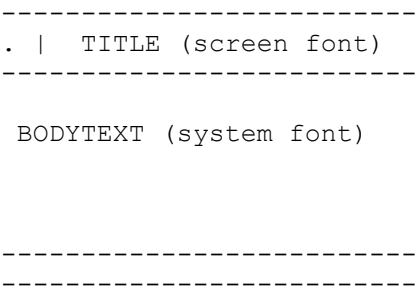
NAME
AllocAsmRequestA -- Alloca ed inizializza un AsmRequest.
AllocAsmRequest -- Varargs stub for AllocAsmRequestA().

SYNOPSIS
areq = AllocAsmRequestA (TagList)
D0 A0

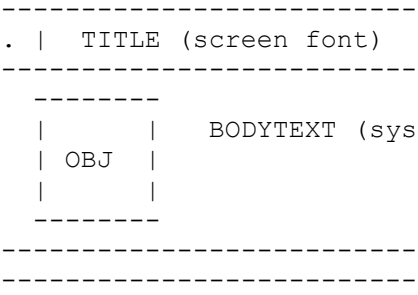
```
struct AsmRequest *AllocAsmRequestA(struct TagItem *);

areq = AllocAsmRequest(firsttag, ...)

struct AsmRequest *AllocAsmRequest(Tag, ...);
```

FUNCTION
Alloca una struttura AsmRequest e la inizializza, facendo riferimento all'array di TagItem passato negli INPUTS. La struttura AsmRequest restituita è READ-ONLY, e deve essere liberata dalla memoria, quando non serve più, utilizzando FreeAsmRequest().
Una volta inizializzata una struttura AsmRequest, questa può essere modificata, permettendo l'uso di diversi tipi di AsmRequest combinati tra loro. In pratica si possono seguire due strade per utilizzare più AsmRequest: inizializzarne il numero che serve, oppure crearne uno solo, e modificarlo, tramite ChangeAsmReqAttrsA(), quando è necessario.

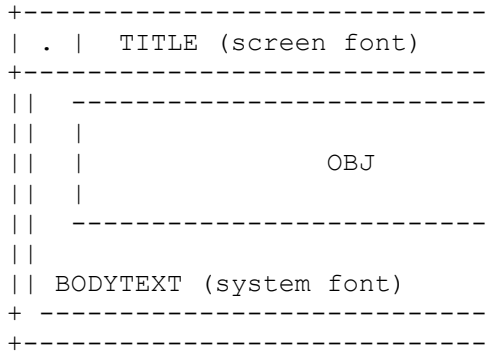
Un'AsmRequest, ha tutte le caratteristiche dell'EasyRequest() della intuition.library, con l'aggiunta di altre speciali funzioni.
Lo schema standar o di default di un AsmRequest, è il seguente:



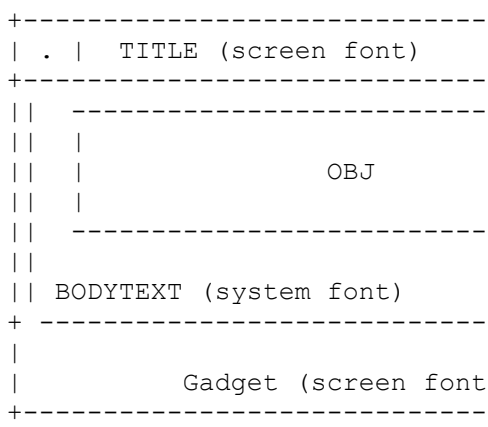
Di default un AsmRequest appare graficamente come i request di sistema.



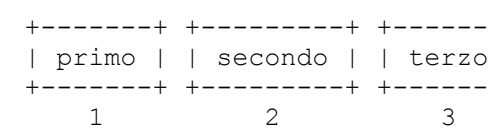
Dov'è Object può essere una qualsiasi immagine in un qualsiasi formato oppure anche un'animazione.
Di object ne sono previsti due attualmente, uno per le immagini o animazioni in qualsiasi formato supportato dalla datatypes.library e uno per i suoni, sempre in qualsiasi formato e sempre supportato dalla datatypes.library.



Posizione che può assumere l'Obejct delle immagini/animazioni.
Questo viene centrato quando il Tag AREQ_TextUnderObject è posto TRUE.



I Gadget vengono addizionati con i font dello schermo.



Il metodo di ordinamento dei pulsanti è identico a quello utilizzato
da intuition per l'EasyRequest(). Stesso dicasi per l'ArgList.

KEYBOARD - Il tasto ESCape provoca per deafult l'uscita dal request
con codice di ritorno NULL. Se sono stati inseriti dei pulsanti
questi possono essere selezionati da tastiera se nella stringa
di inizializzazione era stato inserito il carattere '_' (0x5f)
di sottolineatura. In mancanza di questo carattere può essere
svolta solo la selezione dei pulsanti estremo destro ed
estremo sinistro, tramite la classica combinazione usata anche
nei request di sistema: AmigaLeft+V e AmigaLeft+B, che selezionano
rispettivamente l'estremo sinistro o l'estremo destro.
Nel caso sia stato inserito un solo pulsante, entrambe queste
combinazioni restituiranno un codice di ritorno NULL.

Il tasto RETURN coincide con la combinazione AmigaLeft+V, e
quindi seleziona il pulsante estremo sinistro. Questa chiave può
essere posta in stato OFF (per eventuali motivi di sicurezza) via
tag AREQ_ReturnKey.

INPUTS

TAGS

AREQ_Left (UWORD) - LeftEdge del Request rispetto allo schermo di output.

AREQ_Top (UWORD) - TopEdge del Request rispetto allo schermo di output.

AREQ_REI (struct REI *) - Determina lo schermo di output tramite una struttura REI.

AREQ_Window (struct Window *) - Window di riferimento, questa determina lo schermo di output.

AREQ_Screen (struct Screen *) - Indirizzo dello schermo di output. Questo parametro può essere NULL in questo caso verrà usato lo schermo pubblico di default, o i TAG AREQ_REI, AREQ_Window.

AREQ_Title (STRPTR) - Titolo del AsmRequest. Se NULL, verrà utilizzato il titolo della Window di riferimento se presente, altrimenti si userà 'System Request'. Essendo in una TagList, questo Tag per funzionare correttamente quando posto NULL, deve essere inserito dopo il Tag AREQ_Window. Altrimenti, in modo più corretto, non inserirlo proprio.

AREQ_IDCMP (ULONG) - IDCMPFlags di cui vogliamo essere informati direttamente, e quindi vogliamo controllare noi in modo personale. Per ogni bit ad 1, se viene ricevuto quel messaggio, l'AsmRequest non lo gestisce, ma ci da la possibilità di uscire dall'AsmRequest informandoci del messaggio, o di gestirlo da noi, rimanendo ancora all'interno dell'AsmRequest. Questa scelta dipende se è stato impostato il TAG AREQ_IDCMPHook, che invoca un Hook dall'AsmRequest. Se il TAG AREQ_IDCMPHook è NULL, l'AsmRequest restituisce il codice del flags IDCMP ricevuto. (default to NULL)

AREQ_IDCMPHook (struct Hook *) - Permette di eseguire una routine personale quando uno dei bit settati nel TAG AREQ_IDCMP è verificato. Quest'Hook viene invocato con:

object == (struct REI *) e message == (struct REIMEssage *)

Se l'Hook restituisce (~0) si provoca l'uscita immediata dal request, con RESULT == rim_Class. (default to NULL)

AREQ_LockREI (struct REI *) - Indirizzo della REI che si vuole mettere in stato di attesa, quando il Request è visualizzato. Di solito è la Parant REI, cioè la REI che ha invocato il Request.

AREQ_Justification (UWORD) - Giustificazione del testo, stessi valori usati per i flags di TextFmtRastPortArgs(): ASJ_LEFT, ASJ_RIGHT, ASJ_CENTER.

AREQ_Object (Object *) - Puntatore ad un oggetto boopsi ottenuto con la DataTypes.library. Questo può essere una qualsiasi immagine oppure un'animazione. Le dimensioni vengono richieste direttamente al boopsi il quale viene agganciato alla Window del Request, e ha le sue coordinate a partire dal Left & TopEdge della Window stessa.

AREQ_Sound (Object *) - Puntatore ad una suono aperto via DataTypes. Questo viene attivato all'apertura del request.

AREQ_CenterHScreen (BOOL) - Se TRUE, la Window viene centrata orizzontalmente rispetto allo schermo di output. (Default to FALSE).

AREQ_CenterVScreen (BOOL) - Se TRUE, la Window viene centrata verticalmente rispetto allo schermo di output. (Default to FALSE).

AREQ_CenterMouse (BOOL) - Se TRUE, la Window viene aperta in modo che il puntatore del mouse si trovi al centro della Window stessa. (default to FALSE).

AREQ_TextUnderObject (BOOL) - Se TRUE, il testo verrà visualizzato sotto l'Object (se presente), altrimenti (FALSE) il testo viene posto accanto all'Object. (default to FALSE).

AREQ_APenPattern (UWORD) - Penna di primo piano per il default pattern. O per un simple BackFill quando il request è in modalità

NewLook. (Default to 2).
AREQ_BPenPattern (UWORD) - Penna di secondo piano per il default pattern. O per un simple BackFill quando il request è in modalità NewLook. (Default to 0).
AREQ_PubScreenName (STRPTR) - Nome dello schermo di output. Questo parametro può essere NULL in questo caso verrà usato lo schermo pubblico di default, o i TAG AREQ_REI, AREQ_Window o AREQ_Screen.
AREQ_NewLookBackFill (BOOL) - Rende il Request SENZA la standar retinatura dei requester di sistema, attribuendogli un nuovo look grafico. Se questo flags viene posto TRUE i Tags AREQ_AREQ_APenPattern e AREQ_BREQ_APenPattern vengono usati per un simple backfill superiore ed inferiore. AREQ_AREQ_APenPattern è la venna di background per la parte relativa al body text, AREQ_BREQ_APenPattern è la penna di back ground per la parte relativa ai pulsanti di scelta. (default to FLASE).
AREQ_ReturnKey (BOOL) - Se FALSE, disattiva il riconoscimento del tasto RETURN, lasciando il resto invariato. (default to TRUE).
AREQ_FrameOnly (BOOL) - Se TRUE, elimina il titolo, drag e gadgets dalla Window dell'AsmRequest, che non può essere quindi più spostata o chiusa via mouse. Se FALSE, li rimposta. (default to FALSE).
AREQ_WindowFlags (ULONG) - Flags aggiuntivi per la Window dell'AsmRequest, ad uso personale, USARE CON CAUTELA. (default to NULL).
AREQ_ButtomHook (struct Hook *) - Permette di eseguire una routine personale quando uno dei pulsanti (Buttom) del Request viene rilasciato.

Quest'Hook viene invocato con:

object == (struct REI *) e message == (struct REIMEssage *)

Il campo (ULONG) rim_REICode della struttura REIMessage contiene il numero del pulsante premuto.

Se l'Hook restituisce (~0) si provoca l'uscita immediata dal request, con RESULT == rim_Class. (default to NULL)

RESULT

areq - Se NULL errore, altrimenti ritorna l'indirizzo ad una struttura AsmRequest.

EXAMPLE

NOTES

BUGS

SEE ALSO

AsmRequestArgs(), ChangeAsmReqAttrs(), FreeAsmRequest()
assembly/asmintuition.h

NAME
AllocNewList -- Alloca e inizializza una struttura List.

SYNOPSIS
```
list = AllocNewList()
 D0

struct List *AllocNewList(void);
```

FUNCTION
Alloca una struttura List, e la prepara per essere utilizzata.
Questo comando sarà comodo sia ai programmatori C, sia ai
programmatori Assembly, che troveranno la macro NEWLIST già
all'interno di questo comando.
AllocNewList() è stato scritto per gestire dinamicamente le
liste di sistema, sopratutto, visto che vengono usate dal
LISTVIEW Gadgets della gadtools.library.

INPUTS

RESULT
list - Indirizzo di una struttura List. Se NULL errore.

EXAMPLE

NOTES

BUGS

SEE ALSO
AllocNode(), FreeList(), FreeNode()
exec/lists.i exec/lists.h

NAME
AllocNode -- Alloca ed inizializza una struttura Node.

SYNOPSIS
```
node = AllocNode(list, string, type, pri)
 D0               A0     A1     D0   D1

struct Node *AllocNode(struct List *, STRPTR, UBYTE, BYTE);
```

FUNCTION
Alloca ed inizializza una struttura Node; se list != NULL, il nodo viene inserito nella lista tenendo in considerazione il parametro pri (LN_PRI). L'inserimento avviene utilizzando la MACRO ENQUEUE definita in assembly/asmmacros.i. Questa MACRO svolge lo stesso lavoro di exec/Enqueue().
Se list == NULL, il nodo viene solo allocato, e nessun inserimento nella lista viene svolto; in pratica viene lasciato al TASK il compito di inserire come meglio crede il nodo nella lista.

INPUTS
list - Indirizzo di una struttura List. Se NULL il nodo verrà solo allocato e NON inserito nella lista.
string - Indirizzo da porre in LN_NAME. Può essere NULL.
type - Valore da porre in LN_NAME. Può essere NULL.
pri - Priorità da porre in LN_PRI. Può essere NULL. Questo parametro viene tenuto in considerazione solo quando list != NULL, e permette un inserimento basato sulla priorità dei nodi. Vedi anche exec/Enqueue().

RESULT
node - Indirizzo della struttura Node allocata. Se NULL errore.

EXAMPLE
```
/* Questo è un semplice utilizzo di AllocNewList() e AllocNode() */
/* per creare una lista di labels per un LISTVIEW, sfruttando e  */
/* non sfruttando il parametro priorità (pri) */

struct List *list;
struct Node *node;

      list = AllocNewList();        /* Creo la lista */

      node = AllocNode(list, "Last Element",NULL, -5);
      node = AllocNode(list, "I Element",NULL, 1);
      node = AllocNode(list, "II Element",NULL, 2);
      node = AllocNode(list, "IV Element",NULL, 4);
/*** questo lo gestisco io ***/
      node = AllocNode(NULL, "???? Element",NULL, 0);
/** qui si può chiamare la funzione che più ci aggrada **/
/** il nodo è stato comunque allocato, ma non inserito **/
/** nella lista... ad esempio si può usare AddTail() o **/
/** AddHead(), etc... anche procedure personali... **/

      node = AllocNode(list, "III Element",NULL, 3);

/* etc... come si vede la priorità decide la posizione del node */
/* nella lista, sempre che list != NULL */
```

NOTES

BUGS

```
SEE ALSO
   AllocNewList(), FreeList(), FreeNode()
   exec/AddHead(), AddTail(), Enqueue(), Insert()
   exec/Remove(), RemHead(), RemTail()
   exec/lists.i    exec/lists.h
```

NAME
AllocRastPort -- Alloca ed inizializza una struttura RastPort.

SYNOPSIS
rp = AllocRastPort()
D0

struct RastPort *AllocRastPort(void);

FUNCTION
Alloca una struttura RastPort, e la inizializza tramite il comando graphics.library/InitRastPort(). L'allocazione viene fatta in memoria pubblica, e nessuna struttura BitMap viene attaccata a questa RastPort Usare exec.library/FreeVec() per liberare la memoria.

INPUTS

RESULT
rp - Se NULL errore, altrimenti è l'indirizzo della struttura RastPort allocata.

EXAMPLE

NOTES

BUGS

SEE ALSO
CloneRastPort()
exec.library/FreeVec()

NAME
AS_MenuAddress -- Trova l'indirizzo di un Menu/Item/Sub in una REI.

SYNOPSIS
```
address = AS_MenuAddress (REI, nMenu, nItem, nSubItem)
   D0                     A0     D0     D1      D2

APTR AS_MenuAddress (struct REI *, WORD, WORD, WORD );
```

FUNCTION
Svolge la stessa funzione di intuition.library/ItemAddress() con la sola differanza che questa funzione è più veloce, è legata al sistema REI, e non richiede un pack-code per la ricerca dei Menu, Item o Sub-Item.
Questa ultima caratteristica, lo rende versatile per ricercare lo stato di più Item, come nel caso di Item a mutua-esclusione o con l'opzione Check.

INPUTS
REI - Indirizzo di una struttura REI.
nMenu - Numero Menu, con 0 il primo.
nItem - Numero Item, con 0 il primo. Se -1 (~0) viene restituito l'indirizzo del Menu solamente.
nSubItem - Numero SubItem, con 0 il primo. Se -1 (~0) viene restituito l'indirizzo dell'Item solamente.

RESULT
address - Indirizzo del Menu/Item/SubItem richiesto. Se NULL, errore l'Item/SubItem non è stato trovato.

EXAMPLE

NOTES

BUGS

SEE ALSO
intuition.library/ItemAddress()

NAME
AsmRequestArgs -- Visualizza un AsmRequest.
AsmRequest -- Varargs stub for AsmRequestArgs().

SYNOPSIS
num = AsmRequestArgs(areq, TextFmt, GadgetFmt, ArgList)
D0 A0 A1 A2 A3

LONG AsmRequestArgs(struct AsmRequest *, STRPTR, STRPTR, APTR);

num = AsmRequest(areq, TextFmt, GadgetFmt, Arg1, Arg2, ...);

LONG AsmRequest(struct AsmRequest *, STRPTR, STRPTR, ...);

FUNCTION
Visualizza un request precedentemente inizializzato tramite
AllocAsmRequestA().

INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO

NAME
ChangeAsmReqAttrsA -- Modifica gli attributi di un Request.
ChangeAsmReqAttrs -- Varargs stub for ChangeAsmReqAttrsA().

SYNOPSIS
```
ChangeAsmReqAttrsA(areq, attrs)
                   A0      A1

void ChangeAsmReqAttrsA(struct AsmRequest *, struct TagItem *);

ChangeAsmReqAttrs(areq, firsttag, ...)

void ChangeAsmReqAttrs(struct AsmRequest *, Tag, ...);
```

FUNCTION
Cambia gli attributi di un AsmRequest già allocato.

INPUTS
areq - Indirizzo di una struttura AsmRequest, allocata da
AllocAsmRequestA().
attrs - Un arrary di TagItem che descrivono gli attributi da
modificare. Attualmente sono gli stessi di AllocAsmRequestA(), tutti
quindi possono essere modificati.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
AllocAsmRequestA()
assembly/asmintuition.h

NAME
ChangeChar -- Cambia una carattere con un'altro.

SYNOPSIS
```
nreplace = ChangeChar(string, len , fchar, rchar)
   D0                 A0   D0:16  D1:8   D2:8

ULONG ChangeChar(STRPTR, WORD, BYTE, BYTE);
```

FUNCTION
Cerca un carattere in una sequenza di byte, quando lo ha trovato lo sostituisce, fino a quando non incontra il codice ASCII 0. Ciò è vero se non è stata passata nessuna lunghezza in len, altrimenti si può forzare la scansione per un numero determinato di byte.

INPUTS
string - Puntatore ad una stringa, non necessariamente terminante con NULL.
len - Numero di caratteri da controllare. Se NULL, il controllo avrà termine quando il codice ASCII 0 è trovato.
fchar - Codice ASCII da trovare (Find). QUESTO DEVE ESSERE DIVERSO DA ZERO SE len=NULL.
rchar - Codice ASCII da sostituire.

RESULT
nreplace - Numero di sostituzioni fatte.

EXAMPLE

NOTES
Se come fchar si inserisce il codice ASCII = 0 (zero) e allo stesso tempo si pone len = NULL, il comando non effettuerà sostituzioni. Bisogna inserire in questo caso necessariamente una lunghezza o usare prima il comando FilterChars().

BUGS

SEE ALSO
FilterChars()

NAME
CheckFile -- Controlla lo stato di un file.

SYNOPSIS
```
error = CheckFile (filename, buffer)
 D0                A0       A1

ULONG *CheckFile ( STRPTR, APTR);
```

FUNCTION
Permette di controllare lo stato di un file. Tale file viene aperto in MODE_OLDFILE, se tutto va bene, restituisce NULL come RESULT altrimenti il codice di ritorno di dos.library/IoErr().

CheckFile(), oltre a restituire il codice d'errore, permette di associare già un testo, chiamando dos.library/Fault().
Tale operazione è permessa con l'introduzione di un buffer, che verrà riempito con il testo dell'errore eventualmente associato.
Comunque, bisognerà sempre fare riferimento al codice d'errore di ritorno, per capire se c'è stato o meno un errore.

Il puntatore al buffer deve essere minimo di 80 caratteri. La stringa verrà NULL terminata. Il puntatore al buffer, non deve essere necessariamente introdotto, se lasciato a zero, non provoca nessun effetto, sarà il TASK a preoccuparsi di mostrare sotto forma di testo l'eventuale errore verificatosi.

Il testo è collegato al sistema LOCALE, quindi la stringa errore è nella lingua scelta nelle preferences di sistema.

INPUTS
filename - Puntatore ad una stringa contenente la path e il nome del file da controllare.
buffer - PARAMETRO NON Necessario. Indirizzo di un buffer, min 80 chars, che conterrà la stringa che descrive l'errore nella lingua di sistema.

RESULT
error - NULL se non si è verificato nessun errore. Altrimenti viene restituito un codice d'errore, tramite dos.library/IoErr().

EXAMPLE

NOTES

BUGS

SEE ALSO
dos.library/IoErr(), Fault(), PrintFault()

NAME
CheckSum -- Calcola vari tipi di CheckSum.

SYNOPSIS
```
CheckSum (buffer, type)
            A0     D0

void CheckSum (APTR, ULONG);
```

FUNCTION
Questo comando calcola il CheckSum dei dati presenti nel buffer passato. Per ora il comando supporta due tipi diversi di CheckSum: BootBlock e DataBlock. Una volta calcolato il CheckSum, questo viene automaticamente inserito nel Buffer, all'apposito offset.

INPUTS
buffer - Puntatore al buffer contenente i dati.
type - Specifica il tipo di CheckSum che dev'essere calcolato, per ora, questi sono i tipi definiti:

TCS_BOOTBLOCK - BootBlock.
TCS_DATABLOCK - DataBlock.
TCS_FILEBLOCK - FileBlock.

RESULT

EXAMPLE

ASSEMBLY NOTES
Questo comando non cancella nessun registro, restituisce intatti anche A0 e D0.

BUGS

SEE ALSO
assembly/asmdos.h

NAME
CloneRastPort -- Fa una copia di una struttura RastPort.

SYNOPSIS
clonerp = CloneRastPort (rp)
D0 A0

struct RastPort *CloneRastPort(struct RastPort *);

FUNCTION
Esegue una copia di una struttura RastPort. In pratica alloca una RastPort e copia i dati presenti nella RastPort sorgente, in quelli della copia RastPort.
Usare exec.library/FreeVec() per liberare la memoria.

INPUTS
rp = Indirizzo della struttura RastPort che deve essere clonata.

RESULT
clonerp = Indirizzo della nuova struttura RastPort, identica come dati a quella passata negli INPUTS. Se NULL, errore.

EXAMPLE

NOTES

BUGS

SEE ALSO
AllocRastPort()
exec.library/FreeVec()

NAME
CloseInterface -- Chiude un'interfaccia aperta con OpenInterface().

SYNOPSIS
CloseInterface(i)
A0

```
void CloseInterface(struct Interface *);
```

FUNCTION
Libera tutte le risorse allocate da OpenInterface().

INPUTS
i - Puntatore ad una struttura Interface, ritorno di OpenInterface().

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
CloseREI(), OpenInterface()

NAME
CloseREI -- Chiude una REI aperta con OpenREIA().

SYNOPSIS
```
rei = CloseREI(rei, name)
D0             A0    A1

struct REI *CloseREI(struct REI *, STRPTR );
```

FUNCTION
Chiude una specifica REI, individuata o dall'indirizzo della REI che si vuole chiudere, oppure, se il parametro rei è nullo, tramite il suo nome, cioè una stringa terminata nulla, che contiene un nome che identifica la REI.
E' ovvio che entrambi i parametri NON possono essere NULLI, altrimenti questa funzione fallisce, restituendo un codice di ritorno NULL.

CloseREI() NON libera la memoria dalla struttura REI, si limita a cessare la visualizzazione di una determinata REI. Per riprendere nuovamente la visualizzazione di una REI, basta semplicemente richiamare OpenREIA() passandogli l'indirizzo restituito da CloseREI().

INPUTS
rei - Puntatore ad una struttura REI attualmente visualizzata.
Questo parametro può essere NULL, in tal caso il parametro name sarà utilizzato per cercare la REI nella lista di sistema.
name - Puntatore ad una stringa terminata nulla, che contiene il nome della REI che si desidera occultare. Questo parametro viene tenuto in considerazione SOLO quando rei == NULL.

RESULT
rei - Indirizzo della REI occultata, altrimenti è NULL se la REI non è stata trovata o non è stato possibile occultarla correttamente.

EXAMPLE

NOTES

BUGS

SEE ALSO
FindREI(), OpenREIA()

NAME
CopperEnd -- Termina una UCopList.

SYNOPSIS
CopperEnd (coplist)
A0

void CopperEnd (struct UCopList *);

FUNCTION
Questo comando conclude una lista crescente d'istruzioni Copper definite dal programmatore. CopperEnd() chiama la routine CWait() della graphics. Quando CopperEnd() restituisce il controllo il Task dispone di una lista d'istruzioni Copper completamente definita.

INPUTS
coplist - Indirizzo della struttura UCopList.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
graphics.library/CMove(), CWait()
CopperMove(), CopperWait()

NAME
CopperMove -- Aggiunge un'instruzione alla lista Copper.

SYNOPSIS
CopperMove (coplist,haddress,value)
A0 D0 D1

```
void CopperMove ( struct UCopList *, LONG, LONG );
```

FUNCTION
Aggiunge un'istruzione Copper per spostare il volore 'value' nel registro hardware 'haddress'.

INPUTS
coplist - Indirizzo della struttura UCopList.
haddress - Numero del registro hardware.
value - Valore a 16 bit che dev'essere scritto in tale resgistro.

RESULT

EXAMPLE

NOTES
Il Copper puo' controllare i seguenti gruppi di registri:
1) Qualsiasi registro dall'indirizzo 0x80 esadecimale in su.
2) Qualsiasi registro il cui indirizzo sia compreso tra l'esadecimale 0x40 e 0x80 quando il bit Danger del Copper e' impostato a 1.
Per maggiori dettagli, consultare l'Hardware reference manual.

BUGS

SEE ALSO
graphics.library/CMove(), CWait()
CopperEnd(), CopperWait()

NAME
CopperWait -- Aggiunge un'instruzione alla lista Copper.

SYNOPSIS
```
CopperWait (coplist,vbpos,hbpos)
               A0     D0    D1

void CopperWait ( struct UCopList *, LONG, LONG );
```

FUNCTION
Aggiunge un'istruzione WAIT alla lista Copper. Questo attente il raggiungimento della posizione verticale vbpos e orizzontale hbpos del pennello elettronico di scansione del video per la generazione dell'immagine.

INPUTS
coplist - Indirizzo della struttura UCopList.
vbpos - Posizione verticale del pennello elettronico di scansione del video relativa alla parte della viewport, misurata in pixel.
hbpos - Posizione orizzontale del pennello elettronico di scansione del video relativa alla parte sinistra della viewport, misurata in pixel.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
graphics.library/CMove(), CWait()
CopperMove(), CopperEnd()

NAME
CurrentAppDir -- Fa un lock sulla directory dell'applicazione.

SYNOPSIS
oldLock = CurrentAppDir()
D0

BPTR CurrentAppDir();

FUNCTION
Sposta la directiry corrente, sul quella dell'applicazione. Tutte
le operazioni vengono quindi fatte relative a questa.
Il lock precedente viene restituito per essere repristinato per
altre operazioni sulla root.
Se ad esempio un'applicazione viene lanciata da work:paint/2d/, per
ottenere il lock su questa directory, basta invocare CurrentAppDir().

INPUTS

RESULT
oldLock - Puntatore BCPL al vecchio lock corrente.

EXAMPLE
/* Read a simple File */

```
BPTR oldlock = CurrentAppDir();

      APTR mydata = Load("app.data",NULL,MEMF_PUBLIC);

      /* Other operation */

CurrentDir(oldlock);
```

NOTES

BUGS

SEE ALSO
dos.library/CurrentDir(), Lock(), UnLock()

NAME
DrawBox -- Disegna un rettangolo su una RastPort.

SYNOPSIS
```
DrawBox(rp, left,  top,   width, height)
        A1  D0:16  D1:16  D2:16   D3:16

void DrawBox(struct RastPort *, WORD, WORD, WORD, WORD);
```

FUNCTION
Disegna im modo estremamente rapido un rettangolo su una RastPort.
Molto utile per i programmatori C.

INPUTS
rp - Indirizzo di una struttura RastPort.
left - Coordinata x dell'angolo superiore sinistro.
top - Coordinata y dell'angolo superiore sinistro.
width - Larghezza del rettangolo.
height - Altezza del rettangolo.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO

NAME
DrawFrameStateA -- Disegna una oggetto "frameiclass" su una REI.
DrawFrameState -- Varargs stub for DrawFrameStateA().

SYNOPSIS
DrawFrameStateA(rei, left, top, width, height, state, taglist)
A0 D0 D1 D2 D3 D4 A1

```
void DrawFrameStateA(struct REI *, UWORD, UWORD, UWORD, UWORD,
                     ULONG, struct TagItem *);

DrawFrameState(rei, left, top, width, height, state, firsttag, ...)


void DrawFrameState(struct REI *, UWORD, UWORD, UWORD, UWORD,
                     ULONG, Tag, ...);
```

FUNCTION
Disegna su una REI un oggetto (boopsi) "frameiclass" seguendo una serie di attributi passati in una TagItem list.
DrawFrameStateA() esegue in pratica la stessa funzione del comando DrawBevelBoxA() della gadtools.library, ma in una forma più completa.

Il parametro state permette di cambiare a parità d'immagine, lo stato di disegno, cioè il modo in cui verrà tracciato l'oggetto:

```
IDS_NORMAL              - like DrawImage()
IDS_SELECTED            - represents the "selected state" of a Gadget
IDS_DISABLED            - the "ghosted state" of a gadget
IDS_BUSY            - for future functionality
IDS_INDETERMINATE       - for future functionality
IDS_INACTIVENORMAL      - for gadgets in window border
IDS_INACTIVESELECTED    - for gadgets in window border
IDS_INACTIVEDISABLED    - for gadgets in window border
```

INPUTS
rei - Puntatore ad una struttura REI.
left - Coordinata x dell'angolo superiore sinistro dell'oggetto.
top - Coordinata y dell'angolo superiore sinistro dell'oggetto.
width - Larghezza dell'oggetto.
height - Altezza dell'oggetto.
state - Stato di disegno dell'oggetto.

TAGS
Sono accettati tutti i TAG per la classe "frameiclass", per i nomi e dettagli vedi intuition/imageclass.h.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
intuition/imageclass.h.

NAME
EraseInternalRect -- Pulisce l'interno della Window, usando il
corrente BackFill Hook.

SYNOPSIS
EraseInternalRect(window)
A0

```
void EraseInternalRect(struct Window *);
```

FUNCTION
Pulisce il rettangolo interno ad una Window, senza cancellare i bordi
quindi, utilizzando il corrente BackFill Hook, installato sulla
Window stessa.

INPUTS
window - Puntatore ad una struttura Window.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
graphics.library/EraseRect()

NAME
FileInfo -- Trova le informazioni di un file.

SYNOPSIS
FileInfoBlock = FileInfo(filename)
D0 A0

```
struct FileInfoBlock *FileInfo(STRPTR);
```

FUNCTION
Questo comando alloca in memoria una struttura FileInfoBlock, e la riempie con le informazioni relative al filename passato negli INPUTS. La struttura che si ottiene è una struttura standar FIB della dos, questa va liberata tramite FreeVec().

INPUTS
filename - Nome del file da cui si vuole ricevere il FileInfoBlock.

RESULT
FileInfoBlock - Se NULL errore, altrimenti è l'indirizzo di una struttura FileInfoBlock.

EXAMPLE

NOTES

BUGS

SEE ALSO
dos.library/Examine()
exec.library/FreeVec()

NAME
FilterChars -- Sostituisce un RANGE di codici ASCII.

SYNOPSIS
FilterChars(address, len, lo_range, hi_range, Rchar)
A0 D0 D1:8 D2:8 D3:8

void FilterChars(STRPTR, WORD, BYTE, BYTE, BYTE);

FUNCTION
Questo comando svolge la funzione base di ChangeChar(), solo che invece di sostituire un solo carattere, fà riferimento a un range di codici. I carrateri compresi tra lo_range e hi_range, verrano sostituiti TUTTI dal carattere Rchar.

INPUTS
address - Puntatore all'area di memoria da filtrare.
len - Quantità di byte da filtrare.
lo_range - Valore minimo sotto il quale i codici verranno sostituiti.
hi_range - Valore massimo sopra il quale i codici verranno sostituiti.
Rchar - Codice ASCII che verrà messo al posto di quelli fuori dal RANGE.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
ChangeChar()

NAME
FindAsmGadget -- Trova un AsmGadget via nome.

SYNOPSIS
```
asmgad = FindAsmGadget(rei, name)
  D0                   A0    A1

struct AsmGadget *FindAsmGadget(struct REI *, STRPTR);
```

FUNCTION
Trova il puntatore ad una struttura AsmGadget, eseguendo la ricerca via nome, ricercando nelle liste interne.
Questa funzione è stata scritta per semplificare il lavoro a chi programma in C.

INPUTS
rei - Indirizzo della REI che contiene l'AsmGadget da ricercare.
name - Puntatore ad una stringa a terminazione nulla, che indica il nome del Gadget da ricercare.

RESULT
asmgad - Se NULL non esiste nessun AsmGadget in questa REI. Altrimenti si ottiene l'indirizzo della struttura AsmGadget ricercata.

EXAMPLE

NOTES

BUGS

SEE ALSO

NAME
FindREI -- Trova una struttura REI via nome.

SYNOPSIS
```
rei = FindREI(name)
D0             A1

struct REI *FindREI(STRPTR);
```

FUNCTION
Ricerca nelle liste interne, la struttura REI che risponde al nome indicato negli INPUTS. Se non esiste una lista, significa che nessuna interfaccia è stata aperta, in questo caso la funzione fallisce. Se il nome inserito negli INPUTS viene trovato, l'indirizzo della REI viene restituito.

INPUTS
name - Indirizzo del nome della REI ricercata. Questo dev'essere una stringa a terminazione nulla.

RESULT
rei - Se NULL, non è stato possibile trovare la REI, in quanto NON c'è stata corrispondenza nel nome inserito. Altrimenti viene restituito l'indirizzo della REI ricercata.

EXAMPLE
```
/* C Only Examples */

/* other settings... */

      struct Interface *i = OpenInterface("I:test.rei");

      struct REI *myrei = FindREI("Screen n.1");

/* very very simple...  etc... */
```

NOTES

BUGS

SEE ALSO

NAME
FreeAsmRequest -- Libera la memoria da un AsmRequest.

SYNOPSIS
FreeAsmRequest(areq)
A0

```
void FreeAsmRequest(struct AsmRequest *);
```

FUNCTION
Libera tutte le risorse allocate da AllocAsmRequestA().

INPUTS
areq - Indirizzo di una struttura AsmRequest.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
AllocAsmRequestA()

NAME
FreeList -- Libera la memoria da un'intera lista di nodes.

SYNOPSIS
FreeList(list)
A0

```
void FreeList(struct List *);
```

FUNCTION
Rimuove tutti i nodi di una lista, libera la memoria occuapata da questi (CHE DEVONO ESSERE STATI AGGIUNTI DA AllocNode()!) e libera poi la memoria occuapata dalla struttura list.

INPUTS
list - Indirizzo di una inizializzata struttura list da liberare.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
FreeNode(), FreeListName()

NAME
FreeListName -- Libera la memoria da un'intera lista di nodes.

SYNOPSIS
FreeListName(list)
A0

void FreeListName(struct List *);

FUNCTION
FreeListName() è del tutto simile a FreeList(), l'unica differenza è che se il campo LN_NAME nella struttura Node, è diverso da NULL, quell'indirizzo viene liberato supponendo che è stato precedentemente allocato in memoria via exec/AllocVec(). Se il campo LN_NAME è NULL, FreeListName() si comporta in modo identica a FreeList().

INPUTS
list - Indirizzo di una inizializzata struttura list da liberare.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
FreeList(), FreeNode(), FreeNodeName()

NAME
FreeNode -- Rimuove un nodo e libera la memoria occupata da questo.

SYNOPSIS
FreeNode(list, node)
A0 A1

```
void FreeNode(struct List *, struct Node *);
```

FUNCTION
Elimina un nodo da una lista, agganciato da AllocNode(). Il nodo viene rimosso e la memoria da questo occupato viene liberata. Solo un nodo aggiunto tramite AllocNode() deve essere passato a questa funzione.

INPUTS
list - Indirizzo della lista che contiene il nodo.
node - Indirizzo del node che si vuole rimuovere.

RESULT

EXAMPLE

NOTES
Se il nodo vuole essere solo rimosso, usare Remove() o la MACRO REMOVE per i programmatori assembly.
Per trovare un nodo, usare exec.library/FindName().

BUGS

SEE ALSO
FreeList(), FreeNodeName()
exec/Remove()

NAME
FreeNodeName -- Rimuove LN_NAME, e il nodo.

SYNOPSIS
```
FreeNodeName(list, node)
             A0    A1

void FreeNodeName(struct List *, struct Node *);
```

FUNCTION
FreeNodeName() è del tutto simile a FreeNode(), l'unica differenza è che se il campo LN_NAME è diverso da NULL, quell'indirizzo viene liberato supponendo che è stato precedentemente allocato in memoria via exec/AllocVec(). Se il campo LN_NAME è NULL, FreeNodeName() si comporta in modo identica a FreeNode().

INPUTS
list - Indirizzo della lista che contiene il nodo.
node - Indirizzo del node che si vuole rimuovere.

RESULT

EXAMPLE

NOTES
FreeNodeName() assume che il puntatore al nome in LN_NAME sia stato allocato da exec/AllocVec(). In questo modo si cerca di forzare sempre l'utilizzo di exec/AllocVec() al posto del classico exec/AllocMem(), considerando anche l'indubbia facilità d'uso del primo e l'ormai obsoleta funzione del secondo.

BUGS

SEE ALSO
FreeList(), FreeListName(), FreeNode()
exec/Remove(), FreeVec()

NAME
GetAsmGadgetAttr -- Richiede un attributo di un AsmGadget.

SYNOPSIS
value = GetAsmGadgetAttr(rei, AsmGadget, name, attribute)
D0 A0 A1 A2 D0

ULONG GetAsmGadgetAttr(struct REI *, struct AsmGadget *, STRPTR, ULONG);

FUNCTION
Permette di ricevere delle informazioni sugli elementi che compongono un AsmGadget. Questa funzione permette di richiedere solo un attributo alla volta, e non, come avviene ad esempio per GetREIAttrsA(), una serie di attributi. Ciò è stato fatto, seguendo le regole sui boopsi, che ci insegnano che sono molto più numerose le operazioni di scrittura che quelle di lettura come in questo caso. C'è anche da considerare che il comando in questo modo, risulta essere molto più veloce, come esecuzione in cicli di clock.

INPUTS
rei - Puntatore ad una struttura REI.
AsmGadget - Puntatore all'AsmGadget che si vuole modificare. Se questo parametro è NULL, verrà preso in considerazione il nome per rintracciare l'AsmGadget.
name - Puntatore ad una stringa terminante nulla, che indica il nome dell'AsmGadget che si desidera modificare. Questo parametro è tenuto in considerazione solo se AsmGadget == NULL.
attribute - ti_Tag dell'attributo che si vuole leggere.

TAGS
Sono accettati tutti i Tags della gadtools.library, con l'aggiunta di questi:

AGAT_AsmGadgetHook (struct Hook *) - Indirizzo dell'Hook agganciato a questo AsmGadget.

LISTVIEW_KIND (scrolling list):
AGATLV_SelectedNode (struct Node *) - Restituisce l'indirizzo del nodo attualemente selezionato, ritorna NULL se nessun elemento nel LISTVIEW è attualemente selezionato.

RESULT
value - Valore dell'attributo richiesto. *WARNING* Questo RESULT non può essere controllato da eventuali errori. Questo perchè value, ci informa sul contenuto del tag letto, che può essere quindi anche 0 cioè NULL, di conseguenza tutti i valori possibili sono ammessi come RESULT.

EXAMPLE

NOTES

BUGS

SEE ALSO
SetAsmGadgetAttrsA()
<libraries/gadtools.h>

NAME
GetREIAttrsA -- Richiede gli attributi di una REI.
GetREIAttrs -- Varargs stub for GetREIAttrsA().

SYNOPSIS
numProcessed = GetREIAttrsA(rei, name, taglist)
D0 A0 A1 A2

```
ULONG *GetREIAttrsA(struct REI *, STRPTR, struct TagItem *);

numProcessed = GetREIAttrsA(rei, name, firsttag,...)

ULONG *GetREIAttrs(struct REI *, STRPTR, Tag,...);
```

FUNCTION
Permette di ricevere delle informazioni sugli elementi che compongono una REI, in accordo con gli attributi scelti nella taglist. Per ogni item presente nella taglist, ti_Tag rappresenta l'attributo e ti_Data un puntatore ad una long, dove tu vuoi che l'attributo venga messo.

GetREIAttrsA() riconosce tutti gli attributi passati anche a OpenREIA(). Per una sicura revisione del software e per future eventuali modifiche vedere il file assembly/asmintuition.h e controllare quali attributi vengono riconosciuti.

INPUTS
rei - Puntatore ad una struttura REI, precedentemente aperta da disco da OpenInterface(). Questo parametro può essere NULL, in tal caso il parametro name sarà utilizzato per cercare la REI nella lista di sistema.
name - Puntatore ad una stringa terminata nulla, che contiene il nome della REI che si desidera visualizzare. Questo parametro viene tenuto in considerazione SOLO quando rei == NULL.
taglist - Puntatore ad una lista di TagItem.

TAGS
Vedi OpenREIA() per la lista, e il file assembly/asmintuition.h per i parametri che possono essere utilizzati con questo comando.

RESULT
numProcessed - numero di attributi riempiti con successo.

EXAMPLE

WARNING
Il puntatore usato per registrare l'attributo richiesto e posto in ti_Data, *DEVE* PUNTARE AD UNA LONGWORDS. Questo a prescindere dalla grandezza in bit del dato richiesto.

BUGS

SEE ALSO
OpenREIA(), SetREIAttrsA()
assembly/asmintuition.h

NAME
InterfaceInfo -- Mostra informazioni sull'interfaccia.

SYNOPSIS
```
succ = InterfaceInfo(rei)
 D0                  A0

ULONG InterfaceInfo(struct REI *);
```

FUNCTION
Apre un AsmRequest contenente alcune informazioni sull'interfaccia. Le informazioni sono relative all'interfaccia del TASK, quindi dell'applicazione.

INPUTS
rei - Questo di solito è NULL, ma può puntatore ad una struttura REI che viene bloccata durante la visualizzazione delle informazioni.

RESULT
succ - Ritorna TRUE se l'AsmRequest è stato aperto con successo, se NULL non è stato possibile ottenere le informazioni.

EXAMPLE

NOTES

BUGS

SEE ALSO

NAME
LineInput -- Legge un input da tastiera tramite un Handle.

SYNOPSIS
```
len = LineInput (buffer, chars, Handle)
 D0                A0     D0      D1

ULONG LineInput (APTR, UWORD, BPTR);
```

FUNCTION
Questa funzione ottiene un canale di input dall'Handle specificato. Può essere utilizzata, passando una CON, per ricevere un input da tastiera.

INPUTS
buffer - Indirizzo di un buffer che conterra' i dati letti.
chars - Numero massimo di caratteri da leggere.
Handle - Indirizzo di un Handle, aperto da dos.library/Open().

RESULT
len - Numero di caratteri letti.

EXAMPLE

NOTES

BUGS

SEE ALSO
dos.library/Open(), Input()

NAME
Load -- Carica un file da disco, in memoria.

SYNOPSIS
```
buffer = Load (filename, buffer, typeofmem)
  D0              A0       A1       D0

APTR Load (STRPTR, APTR, LONG);
```

FUNCTION
Permette di caricare un qualsiasi file in memoria, sfruttando la funzione dos.library/Open(). Questo comando NON effettua nessuna rilocazione del file.
Il puntatore al filename, è una stringa, completa di path, che indica il file che deve essere caricato in memoria.

Molto interessante è la possibiltà di pre-allocare un buffer proprio invece di farlo allocare dal Load() stesso.
Se infatti negli INPUTS, si passa un buffer, il file verrà caricato dentro questo.
In linea di massima, si consiglia vivamente di passare sempre un buffer NULL, lasciando il compito dell'allocazione a Load(), ciò permetterà infatti di usare exec.library/FreeVec(), quando si dovrà liberare la memoria.

Load() alloca un'intero blocco di memoria, corrispondente alla grandezza in byte del file da caricare. Quindi si consiglia l'uso di questo comando, in tutte quei casi dove è certa la memoria disponibile, e la sua frammentazione.

Il suo utilizzo è indicato per file che non superano i 500K di lunghezza, anche se in sistemi dotati di grandi quantitativi di RAM (Fast o Chip) il suo uso è possibile con qualunque dimensione di file.

INPUTS
filename - Puntatore al nome del file da caricare.
buffer - Puntatore ad un buffer dove verrano caricati i dati.
Se NULL, la routine penserà automaticamente ad allocare un buffer dove caricare i dati; in questo caso si deve SEMPRE specificare anche il tipo di memoria da allocare: MEMF_PUBLIC etc...
typeofmem - Tipo di memoria da allocare. Questo parametro viene tenuto in considerazione, quando buffer = NULL.

RESULT
buffer - Se NULL errore, altrimenti è l'indirizzo del buffer che contiene i dati del filename. Se era stato passato un buffer pre-allocato, e quindi, proprio, questo indirizzo è lo stesso passato negli INPUTS, altrimenti è il buffer allocato dal comando.

EXAMPLE

NOTES

BUGS
Con la memoria troppo frammentata, Load() può fallire, per files troppo ampi.

SEE ALSO
CheckFile(), Save()
dos.library/Open(), Read()

exec.library/FreeVec()

NAME
LockREI -- Mette in WAIT una REI.

SYNOPSIS
```
rei = LockREI (rei, name)
D0             A0    A1

struct REI *LockREI(struct REI *, STRPTR);
```

FUNCTION
Blocca una REI da qualsiasi tentativo di input da parte dell'utente. Questo impedisce di accedere sia ai Menu che ai Gadget che sono eventualmente attaccati a questa REI, sia via Mouse che via tastiera. Il puntatore del Mouse, viene posto in attesa, e il suo disegno viene cambiato con quello di BUSY di sistema.
Questa funzione si assicura che la REI che si vuole bloccare non sia già stata bloccata. Se entrambi gli INPUTS sono NULL, verrà bloccata la REI attualmente attiva, se viene trovata ovviamente.

INPUTS
rei - Indirizzo della REI che si vuole bloccare. Se NULL, verrà preso come riferimento il nome della REI.
name - Puntatore ad una stringa terminata nulla, che contiene il nome della REI che si desidera bloccare. Questo parametro viene tenuto in considerazione SOLO quando rei == NULL.

RESULT
rei - Se NULL, non è stato possibile bloccare la REI. Altrimenti questo è l'indirizzo della struttura REI bloccata.

EXAMPLE

NOTES

BUGS

SEE ALSO
UnlockREI()

NAME
NewAllocRaster -- Rifacimento di AllocRaster().

SYNOPSIS
planeptr = NewAllocRaster (width, height)
D0 D0:16 D1:16

PLANEPTR NewAllocRaster (WORD, WORD);

FUNCTION
Un modo più semplice di allocare un bitplanes già pulito. Questa funzione è più veloce rispetto ad graphics.library/AllocRaster(), in più permette di allocare un plane già pulito, e facilità anche la cancellazione di questo, che può essere mandato direttamente a exec.library/FreeVec().

(V41.1) - Per ragioni di compatibilità futura, si sconsiglia l'uso di questa funzione. Utilizzare sempre, quando è possibile, i comandi AddBitPlanes() e RemoveBitPlanes(); lo stesso discorso vale anche per graphics.library/AllocRaster(), FreeRaster().

INPUTS
width - larghezza in pixel del plane desiderato.
height - altezza in pixel del plane desiderato.

RESULT
planeptr - Se NULL errore, altrimenti rappresenta il puntatore al bitplanes, già pulito.

EXAMPLE

NOTES

BUGS
I planes creati devono essere liberari manualmente, NON SI PUO' USARE LA FUNZIONE RemoveBitPlanes().

SEE ALSO
AddBitPlanes()
exec.library/FreeVec()

NAME
OpenInterface -- Apre una interfaccia da disco.

SYNOPSIS
```
i = OpenInterface(filename)
D0                A0

struct Interface *OpenInterface(STRPTR);
```

FUNCTION
Apre dal device un'interfaccia creata con REI-Editor.
OpenInterface() permette una semplificata gestione nel caricamento di un'interfaccia, infatti il filename passato negli INPUTS può essere anche completo di path, Examples: work:utils/myinter/cogito.rei
In questo caso, tutta la path viene presa in considerazione, e se per qualsiasi motivo OpenInterface() non riesce a caricare l'interfaccia recupera il solo nome del file (Es. cogito.rei) cercandolo prima nella directory dell'applicazione stessa, e se anche qui fallisce, come ultima alternativa, prova nel cassetto SYS:I, che per default e definito come il cassetto dell'interfaccie di sistema.

Per ragioni di compatibilità e facile lettura, la sintassi di un corretto filename è: NOMEAPPLICAZIONE.rei

In linea generale, sapendo la priorità di lettura del file .rei da parte di OpenInterface(), consigliamo di porre in SYS:I le interfaccie originali di tutte le applicazione, e quelle eventualmente da voi personalizzate/modificate nella stessa path dell'applicazione. In questo modo cerchiamo di mantenere una certa compatibilità futura.

INPUTS
filename - Puntatore ad una stringa terminante nulla, che indica il file .rei da aprire.

RESULT
i - Puntatore ad una struttura Interface. Se NULL, il file .rei non è stato trovato, o è stato impossibile caricarlo.

EXAMPLE
```
/* Open and Display a REI */
/* C language version     */

main() {

struct Interface *i;
struct REI *myrei;

      if(i = OpenInterface("MiaApplicazione.rei"))
      {
            if(myrei = OpenREIA(NULL, "main", NULL))
            {
                  /* Use the interface */

                  CloseREI(myrei, NULL);
            }
            CloseInterface(i);
      }
} /* end main */
```

NOTES
Un metodo consigliato per permettere l'uso di più interfaccie, è quello

di scrivere l'applicazione in modo che determini il nome del file .rei direttamente dall'applicazione stessa. Questa può essere fatto via TOOLTYPES se l'applicazione viene lanciata da Workbench, o via parse Shell, se l'applicazione viene lanciata dal CLI.
Ad esempio INTERFACE=cogito2.rei posto nei TOOLTYPES o nel parse permette un facile switching tra un'interfaccia e un'altra, senza complicare molto la vita.

BUGS

SEE ALSO
CloseInterface(), OpenREIA()

NAME
OpenREIA -- Apre sul video una specifica REI.
OpenREI -- Varargs stub for OpenREIA().

SYNOPSIS

```
rei = OpenREIA(rei, name, taglist)
D0             A0    A1     A2

struct REI *OpenREIA(struct REI *, STRPTR, struct TagItem * );

rei = OpenREI(rei, name, firsttag,...)

struct REI *OpenREI(struct REI *, STRPTR, Tag,...);
```

FUNCTION
Apre una specifica REI, individuata o dall'indirizzo della REI che si vuole aprire, oppure, se il parametro rei è nullo, tramite il suo nome, cioè una stringa terminata nulla, che contiene un nome che identifica la REI.
E' ovvio che entrambi i parametri NON possono essere NULL, altrimenti questa funzione fallisce, restituendo un codice di ritorno NULL.
In caso di successo viene restituita la struttura REI:

```
STRUCTURE REI,LN_SIZE                ; (struct Node *)
      APTR  rei_Screen         ; (struct Screen *)
      APTR  rei_Window         ; (struct Window *)
      APTR  rei_Menu           ; (struct Menu *)
      ULONG rei_reserved1            ; future use
      APTR  rei_VI                   ; (struct vi *) visualinfo see above
      STRUCT      rei_reserved2,8*4 ; future implement....
      UWORD rei_ID                   ; REI ID per utente
      APTR  rei_UserData             ; REI Userdata per utente
```

INPUTS
rei - Puntatore ad una struttura REI, precedentemente aperta da disco da OpenInterface(). Questo parametro può essere NULL, in tal caso il parametro name sarà utilizzato per cercare la REI nella lista di sistema.
name - Puntatore ad una stringa terminata nulla, che contiene il nome della REI che si desidera visualizzare. Questo parametro viene tenuto in considerazione SOLO quando rei == NULL.
taglist - Puntatore ad una lista di TagItem.

TAGS
REIT_Screen (struct Screen *) - Indirizzo dello schermo dove la REI deve essere aperta. Se NULL, la REI verrà visualizzata sullo schermo pubblico di default. (dafault to NULL).
REIT_WindowTAG (struct TagItem *) - Indirizzo di una lista di TagItem relativa alla Window, da usare al posto di quella definita con il programma REI-Editor. (default to NULL).
REIT_ScreenTAG (struct TagItem *) - Indirizzo di una list di TagItem relativa allo Screen, da usare al posto di quella definita con il programma REI-Editor. (default to NULL).
REIT_WindowTextAttr (struct TextAttr *) - Tipo di Font per questa la Window di questa REI. Se NULL, come di default, verrà usato il font di sistema. (default to NULL).
REIT_WindowTextFont (struct TextFont *) - Tipo di Font per questa la Window di questa REI. Se NULL, come di default, verrà usato il font di sistema. (default to NULL).
REIT_ScreenFont (BOOL) - Se TRUE, s'impone alla Window di usare gli

stessi font usati dallo schermo su cui si aprirà. Di default la Window usa per la barra i font dello schermo, e per il testo i font di sistema, o i font decisi tramite i TAGS REIT_WindowTextAttr e REIT_WindowTextFont. Se questo TAG è TRUE, s'impone in pratica di usare anche per il testo i font dello schermo. (default to FALSE).

REIT_NewMenu (struct NewMenu *) - Indirizzo ad una struttura NewMenu. Sono i Menu per questa Window. (default to NULL).

REIT_NewMenuTAG (struct TagItem *) - Indirizzo ad una lista di TagItem per i Menu se attaccati via Tag. (dafult to NULL).

REIT_UserData (LONG) - 32bit di data, definibili a piacere. (default to NULL).

REIT_LayoutCallBack (struct Hook *) - Indirizzo ad una struttura Hook per il BackFill della layer di questa Window. Questo Hook verrà chiamato con object == (struct RastPort *) result ->RastPort e message == [(Layer *) layer, (struct Rectangle) bounds, (LONG) offsetx, (LONG) offsety]
per maggiori dettagli, vedi Layers.library/InstallLayerHook().

REIT_CustomHook (struct Hook *) - Indirizzo ad una struttura Hook per eseguire operazioni proprie sulla Window. Questo Hook viene chiamato dopo che la Window della REI è stata aperta e preparata. Questo Hook verrà chiamato con object == (struct Window *) e message == NULL. A6 == AssemblyBase.

REIT_RememberPos (BOOL) - Se TRUE, dopo un CloseREI(), ricorda la posizione (LeftEdge, TopEdge) della Window. (default to FALSE).

REIT_RememberSize (BOOL) - Se TRUE, dopo un CloseREI(), ricorda le dimensioni (Width, Height) della Window. (default to FALSE).

REIT_CenterHScreen (BOOL) - Se TRUE, la Window viene centrata orizzontalmente rispetto allo schermo di output. (Default to FALSE).

REIT_CenterVScreen (BOOL) - Se TRUE, la Window viene centrata verticalmente rispetto allo schermo di output. (Default to FALSE).

REIT_CenterMouse (BOOL) - Se TRUE, la Window viene aperta in modo che il puntatore del mouse si trovi al centro della Window stessa. (default to FALSE).

REIT_NoFontSensitive (BOOL) - Se TRUE, la Window NON ha più le dimensioni relative ai font. USARE CON CAUTELA. (Default to FALSE).

REIT_WindowTitle (STRPTR) - Puntatore ad una stringa terminata nulla, che descrive il titolo della finestra di questa REI. Questo Tag è stato introdotto per facilitare il settaggio del titolo, relativamente alla localizzazione di una REI e per non introdurre una TagList via REIT_WindowTAG per il titolo solamente. (default to NULL).

REIT_Window (struct Window *) - Puntatore ad una struttura Window che decide lo schermo di output di questa REI. (default to NULL).

RESULT

REI - Indirizzo della struttura REI visualizzata. Se NULL Errore. In questo caso, l'intera interfaccia non viene visualizzata.
Se era stato passato un valido parametro rei negli INPUTS, questo valore DEVE corrispondere, cioè viene restituito lo stesso indirizzo.

EXAMPLE

NOTES

BUGS

SEE ALSO

CloseREI(), FindREI(), GetREIAttrsA(), SetREIAttrsA()
assembly/asmintuition.h

NAME
ReAllocVec -- Rialloca un'area di memoria.

SYNOPSIS
```
newmemoryBlock = ReAllocVec(oldmemoryBlock, newsize, newattribute)
      D0                    A0           D0         D1

ULONG *ReAllocVec(ULONG *, ULONG, ULONG);
```

FUNCTION
Permette di modificare dinamicamente la grandezza di un'area di memoria precedentemente allocata via exec/AllocVec(). Il nuovo blocco di memoria può essere più grande o più piccolo di quello precedente, e può avere nuovi attributi di memoria.
Se il parametro newattribute viene lasciato NULL, verranno presi come nuovi attributi quelli del vecchio blocco di memoria.
Se il nuovo blocco di memoria che si vuole allocare è più piccolo di quello precedente, alcuni dati potrebbero essere persi. Questa funzione infatti copia il contenuto di oldmemoryBlock in quello di newmemoryBlock.

INPUTS
oldmemoryBlock - Blocco di memoria che vuole essere modificato.
newsize - Nuova quantità di byte da allocare.
newattribute - Tipo di memoria. Se NULL verrà presa in considerazione il tipo di memoria di oldmemoryBlock.

RESULT
newmemoryBlock - Nuovo indirizzo da usare al posto di oldmemoryBlock. Se NULL non è stato possibile allocare altra memoria.

EXAMPLE

NOTES

BUGS

SEE ALSO

NAME
RefreshREI -- Ridisegna l'intera REI.

SYNOPSIS
RefreshREI (REI)
A0

```
void RefreshREI(struct REI *);
```

FUNCTION
Ridisegna tutti i Gadget attaccati ad una REI.
Repristina tutte le informazioni per la gadtools.
Ridisegna tutta la Window, i system Gadgets e i Bordi.

INPUTS
REI - Indirizzo della struttura REI da ridisegnare.

RESULT
L'intera REI viene ridisegnata ed aggiornata.

EXAMPLE

NOTES
Questo comando può aumentare le suo funzioni di refresh in versioni future.

BUGS

SEE ALSO
gadtools.library/GT_RefreshWindow()
intuition.library/RefreshGList()

NAME
RemoveBitPlanes -- Rimuove un certo numero di planes da una BitMap.

SYNOPSIS
```
num = RemoveBitPlanes (bm, num)
D0                     A0  D0

ULONG RemoveBitPlanes (struct BitMap *, WORD);
```

FUNCTION
Permette di rimuovere un certo numero di planes da una struttura BitMap. Questi vengono rimossi fisicamente dalla struttura BitMap, e la memoria da loro occupata viene liberata. Per ogni planes rimosso il campo bm_Depth della BitMap, viene decrementato, e il puntatore al planes viene pulito.

INPUTS
bm - Indirizzo di una struttuta BitMap inizializzata.
num - Numero di bitplanes da rimuovere. Questo può essere uguale al totale dei plane presenti in una BitMap, quindi tutti i planes possono essere rimossi. L'importante è che num sia uguale o maggiore al contenuto del campo bm_Depth, altrimenti questa funzione fallisce.

RESULT
num - Se NULL, nessun planes è stato rimosso, o il numero di planes da rimuovere era troppo elevato rispetto al contenuto del campo bm_Depth. Altrimenti indica il numero dei planes che sono stati rimossi.

EXAMPLE

NOTES

BUGS
Questa funzione per ora non utilizza graphics.library/GetBitMapAttr() sarà aggiornata in future versioni della libreria.

SEE ALSO
AddBitPlanes()
graphics.library/FreeBitMap(), FreeRaster(), GetBitMapAttr()

NAME
RevertMem -- Inverte il contenuto di un'area di memoria.

SYNOPSIS
RevertMem (start, end, type)
A0 A1 D0:8

void RevertMem (APTR, APTR, UBYTE);

FUNCTION
Questa funzione permette d'invertire il contenuto di un'area di memoria, definita da start ed end. In pratica inverte la sequenza con cui i byte si susseguono nell'area di memoria. Tale inversione può essere svolta su byte, su word, o su long. Questo viene deciso dal parametro type, che indica il numero di bit su cui si vuole operare. Il primo byte/word/long diventa quindi l'ultimo byte/word/long, il secondo byte/word/long diventa il penultimo byte/word/long, etc.

INPUTS
start - Inizio dell'area di memoria da invertire.
end - Fine dell'area di memoria da invertire. Questo può essere calcolato, aggiungengo a start il numero di byte su cui si vuole operare, a prescindere se poi l'operazione d'inversione avverrà su byte, word o long.
type - Indica il numero di bit (8, 16 o 32) su cui operare. I valori ammessi sono:
TYPEF_BYTE - opera su byte
TYPEF_WORD - opera su word
TYPEF_LONG - opera su long

RESULT

EXAMPLE

```
**
** Ruota 2 word all'indirizzo:
**
** test:    dc.b "ABCD"
**

.................
move.l      assemblybase,a6
lea   test,a0
lea   4(a0),a1
moveq #TYPEF_WORD,d0
JSR   _LVORevertMem(a6)
.............

**
** Dopo la chiamata a RevertMem() si avrà:
**
** test:    dc.b "CDAB"
**
```

NOTES
Come si vede dall'EXAMPLE, l'indirizzo end, viene calcolato con molta facilità, partendo dall'indirizzo start. Comunque si tenga presente che la differanza tra l'indirizzo end e l'indirizzo start, deve sempre essere uguale alla quantità di bye su cui si deve operare.

BUGS

SEE ALSO

NAME
Save -- Scrive un buffer su device.

SYNOPSIS
```
len = Save (filename, buffer,len)
 D0            A0       A1   D0

ULONG Save (STRPTR, APTR, ULONG);
```

FUNCTION
Apre un canale di output su un device. Opera in pratica in modo inverso a Load().
Questo comando è stato concepito per la registrazione di un file su disco, ma il suo output può essere re-indirizzato su tutte le periferiche (device) disponibili.

L'apertura del file viene effettuata in MODE_READWRITE, quindi se il filename è già presente sulla periferica, questo verrà sovrascritto senza alcun preavviso.
Il compito di controllare la eventuale già presenza del file, è lasciato al TASK, per questo può essere usato CheckFile().

INPUTS
filename - Indirizzo di una stringa, contenente il nome del file che si vuole scrivere. Questo può essere preceduto da RAM:, PRT:, CON: etc...
buffer - Puntatore al buffer dati. Questi sono i dati che verrano scritti sulla periferica.
len - Numero di byte da trasferire su periferica. Questo parametro può essere NULL quando il buffer è stato acquisito via Load(), e a quest'ultimo è stato lasciato il compito di allocare il buffer. Solo in questo caso, len può essere NULL, altrimenti dovrà essere specificata.

RESULT
len - Quantità di byte scritti. Se NULL errore.

EXAMPLE
```
/* Load and Save example.c */

APTR mybuffer = Load("Work:page.txt",NULL,MEMF_PUBLIC);

      /* Your READ or WRITE operation */

ULONG len = Save("Work:page.txt",mybuffer,NULL);
```

NOTES

BUGS

SEE ALSO
CheckFile(), Load()

NAME
SetAsmGadgetAttrsA --Cambia gli attributi di un AsmGadget.
SetAsmGadgetAttrs -- Varargs stub for SetAsmGadgetAttrsA().

SYNOPSIS
```
success = SetAsmGadgetAttrsA(rei, AsmGadget, name, taglist)
  D0                         A0      A1       A2     A3

BOOL SetAsmGadgetAttrsA(struct REI *,struct AsmGadget *,STRPTR,
                            struct TagItem *);

success = SetAsmGadgetAttrs(rei, AsmGadget, name, firsttag,...)

BOOL SetAsmGadgetAttrs(struct REI *,struct AsmGadget *,STRPTR,Tag,...);
```

FUNCTION

INPUTS
rei - Puntatore ad una struttura REI.
AsmGadget - Puntatore all'AsmGadget che si vuole modificare. Se questo parametro è NULL, verrà preso in considerazione il nome per rintracciare l'AsmGadget.
name - Puntatore ad una stringa terminante nulla, che indica il nome dell'AsmGadget che si desidera modificare. Questo parametro è tenuto in considerazione solo se AsmGadget == NULL.
taglist - Puntatore ad una lista di TagItems.

TAGS
Sono accettati tutti i Tags della gadtools.library, con l'aggiunta di questi:

SGAT_AsmGadgetHook (struct Hook *) - Aggancia un nuovo Hook o NULL, al Gadget in questione. Quando questo Gadget verrà rilasciato, WaitREIMsg() invocherà questo Hook. Si può passare anche un NULL, in questo caso nessun Hook viene agganciato al Gadget, e quando il gadget viene rilasciato, WaitREIMsg() restituisce il controllo al TASK.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
GetAsmGadgetAttr(), SetREIAttrsA()
<libraries/gadtools.h>

NAME
SetREIAttrsA -- Imposta gli attributi di una REI.
SetREIAttrs -- Varargs stub for SetREIAttrsA().

SYNOPSIS
succ = SetREIAttrsA(rei, name, taglist)
D0 A0 A1 A2

BOOL SetREIAttrsA(struct REI *, STRPTR, struct TagItem *);

succ = SetREIAttrsA(rei, name, firsttag,...)

BOOL SetREIAttrs(struct REI *, STRPTR, Tag,...);

FUNCTION
Permette di impostare in modo dinamico, gli attributi di una REI, in accordo con gli attributi scelti nella taglist.
SetREIAttrsA() modifica in tempo reale, le caratteristiche di una REI visualizzata. Questo comando dunque va usato *SOLO* su REI che sono aperte sul video, e non su eventuali REI chiuse o ancora da aprire.

SetREIAttrsA() riconosce tutti gli attributi passati anche a OpenREIA(). Per una sicura revisione del software e per future eventuali modifiche vedere il file assembly/asmintuition.h e controllare quali attributi vengono riconosciuti.

INPUTS
rei - Puntatore ad una struttura REI, precedentemente aperta da disco da OpenInterface(). Questo parametro può essere NULL, in tal caso il parametro name sarà utilizzato per cercare la REI nella lista di sistema.
name - Puntatore ad una stringa terminata nulla, che contiene il nome della REI che si desidera visualizzare. Questo parametro viene tenuto in considerazione SOLO quando rei == NULL.
taglist - Puntatore ad una lista di TagItem.

TAGS
Vedi OpenREIA() per la lista, e il file assembly/asmintuition.h per i parametri che possono essere utilizzati con questo comando.

RESULT
succ - Se FALSE non è stato possibile impostare qualche parametro. Altrimenti ritorna sempre TRUE, se tutto è andato bene.

EXAMPLE

NOTES

BUGS

SEE ALSO
OpenREIA(), GetREIAttrsA()
assembly/asmintuition.h

NAME
StringBinToValue -- Converte stringa binaria in valore.

SYNOPSIS
```
value = StringBinToValue(string, optlen)
 D0                      A0      D0

LONG StringBinToValue(STRPTR, WORD);
```

FUNCTION
Questa routine converte una stringa binaria, cioe' una serie di 1 e 0 in valore decimale LONG. In parole povere opera in maniera opposta al comando ValueToStringBin() presente nell' assembly.library. StringBinToValue() converte una stringa di lunghezza variabile da 1 carattere a 32 caratteri, coprendo quindi il campo dei 32 Bit.

Sono permessi i caratteri:
- '%' - Opzionale, serve solo come indicatore nella stringa, per sottolineare che si tratta di un valore binario. Questo carattere non ha effetto sulla conversione.
- '+' - Sottolinea la positività della conversione, che è già stabilita di default, quindi anche questo non ha effetto sulla conversione.
- '-' - Opera un'operazione di NOT logico sul risultato finale. Indica dunque che si vuole il NOT della stringa passata.

INPUTS
string - Puntatore ad una stringa terminante nulla, che contiene la sequenza ASCII binaria da convertire.
optlen - Questo parametro è di default NULL, altrimenti permette di specificare il numero dei bit su cui operare la conversione. Di default opera su tutti e 32 i bit, quando appunto questo parametro è NULL.

RESULT
value - Valore LONG convertito.

EXAMPLE
```
**
** Tipi di stringa e conversioni:
**

**
** optlen=NULL;
**

string = "000100"    ---> 4
string = "%0100"     ---> 4
string = "-%100"     ---> -5
string = "%-100"     ---> -5
string = "-100"      ---> -5  etc...

**
** optlen=4;
**

string = "1110010"   ---> 2
string = "0010"      ---> 2
string = "111111"    ---> 15
```

NOTES

BUGS

Se viene introdotto il carattere (-), questo ANNULLA sempre e comunque il carattere (+), qualunque sia la posizione.

SEE ALSO

StringDecToValue(), StringHexToValue()
ValueToStringBin()

NAME
StringDecToValue -- Converte stringa decimale in valore.

SYNOPSIS
```
value = StringDecToValue(string)
 D0                      A0

LONG StringDecToValue(STRPTR);
```

FUNCTION
Questa routine converte una stringa decimale in un valore. Il valore che ritorna in value occupa al massimo una LONG (32 Bit).
Il massimo numero convertivile e' quindi $FFFFFFFF=#4294967295.

Sono permessi i caratteri:

- '#' - Opzionale, serve solo come indicatore nella stringa, per sottolineare che si tratta di un valore decimale. Questo carattere non ha effetto sulla conversione.
- '+' - Sottolinea la positività della conversione, che è già stabilita di default, quindi anche questo non ha effetto sulla conversione.
- '-' - Rende negativo il risultato.

INPUTS
string - Puntatore ad una stringa terminante nulla, con la sequenza ASCII del valore decimale da convertire.

RESULT
value - Valore LONG convertito.

EXAMPLE
```
string = "49152"   --> value = 0000C000
string = "#64"     --> value = 00000040
string = "#-64"    --> value = FFFFFFC0
string = "-#+64"   --> value = FFFFFFC0 etc...
```

NOTES

BUGS
Se viene introdotto il carattere (-), questo ANNULLA sempre e comunque il carattere (+).

SEE ALSO
StringBinToValue(), StringHexToValue()
ValueToStringDec()

NAME
StringHexToValue -- Converte stringa esadecimale in valore.

SYNOPSIS
```
value = StringHexToValue(string)
 D0                      A0

LONG StringHexToValue(STRPTR);
```

FUNCTION
Questa routine converte una stringa esadecimale, cioè una stringa contenente un numero rappresentato in notazione esadecimale. Il valore che ritorna in D0 occupa al massimo una LONG (32 Bit) quindi il massimo valore convertibile è: $FFFFFFFF

Sono permessi i caratteri:

- '$' - Opzionale, serve solo come indicatore nella stringa, per sottolineare che si tratta di un valore esadecimale. Questo carattere non ha effetto sulla conversione.
- '+' - Sottolinea la positività della conversione, che è già stabilita di default, quindi anche questo non ha effetto sulla conversione.
- '-' - Rende negativo il risultato.

INPUTS
string - Puntatore ad una stringa terminante nulla, con la sequenza ASCII del valore esadecimale da convertire.

RESULT
value - Valore LONG convertito.

EXAMPLE
```
string = "$C000"   --> value = 0000C000
string = "+Bfe0aA" --> value = 00BFE0AA
string = "$-100"   --> value = FFFFFF00
string = "-100"    --> value = FFFFFF00
string = "-$100"   --> value = FFFFFF00
string = "-+$100"  --> value = FFFFFF00 etc...
```

NOTES

BUGS
Se viene introdotto il carattere (-), questo ANNULLA sempre e comunque il carattere (+).

SEE ALSO
StringBinToValue(), StringDecToValue()
ValueToStringHex()

NAME
StringToLower -- Converte una stringa in lowercase.

SYNOPSIS
StringToLower(locale, string, len)
A0 A1 D0

```
void StringToLower(struct Locale *, STRPTR, WORD);
```

FUNCTION
Converte un'intera stringa ASCII, in lowercase, in accordo con il sistema di localizzazione.

INPUTS
locale - Puntatore ad una struttura Locale, o NULL per la struttura Locale di default.
string - Indirizzo della stringa da convertire. Questa deve terminare NULL, se si vuole passare len = NULL.
len - Numero di caratteri da convertire, se NULL l'intera stringa sarà convertita.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
StringToUpper()
locale.library/ConvToUpper(), ConvToLower()

NAME
StringToUpper -- Converte una stringa in uppercase.

SYNOPSIS
StringToUpper(locale, string, len)
A0 A1 D0

```
void StringToUpper(struct Locale *, STRPTR, WORD);
```

FUNCTION
Converte un'intera stringa ASCII, in uppercase, in accordo con il sistema di localizzazione.

INPUTS
locale - Puntatore ad una struttura Locale, o NULL per la struttura Locale di default.
string - Indirizzo della stringa da convertire. Questa deve terminare NULL, se si vuole passare len = NULL.
len - Numero di caratteri da convertire, se NULL l'intera stringa sarà convertita.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
StringToLower()
locale.library/ConvToUpper(), ConvToLower()

NAME
TextFmtRastPortArgs -- Stampa una 'C'-style String su una RastPort.
TextFmtRastPort -- Varargs stub for TextFmtRastPortArgs().

SYNOPSIS
```
next = TextFmtRastPortArgs(rp, TextFmt, left, top, flags, ArgList)
 D0                        A1    A0      D0    D1    D2      A2

APTR TextFmtRastPortArgs(struct RastPort *, STRPTR, WORD, WORD,
                         UWORD, APTR);

next = TextFmtRastPort(rp, TextFmt, left, top, flags, Arg1, Arg2, ...)

APTR TextFmtRastPort(struct RastPort *, STRPTR, WORD, WORD,
                         UWORD, ...);
```

FUNCTION
Stampa una stringa in formato C-style, su una RastPort.
Questo comando permette di utilizzare il formato stringa utilizzato nello standar printf del linguaggio C.
TextFmtRastPortArgs() utilizza il comando exec.library/RawDoFmt() per formattare la stringa; vedere gli autodoc di exec.library/RawDoFmt().

INPUTS
rp - Puntatore ad una struttura RastPort.
TextFmt - Indirizzo ad una stringa, in formato c-style.
left - Coordinata x dell'inizio testo.
top - Coordinate y dell'inizio testo.
flags - Parametri aggiuntivi del testo da visualizzare:

ASJ_LEFT - Giustifica il testo tutto a sinistra.
ASJ_RIGHT - Giustifica il testo tutto a destra.
ASJ_CENTER - Centra il testo.

Se NULL viene usato come default flags ASJ_LEFT. In questa versione sono supportati solo i flags di giustificazione.

ArgList - Array degli argomenti per il comandi do formattazione.

RESULT
next - Ritorna un puntatore alla fine del DataStream appena processato per permettere di utilizzare lo stesso DataStream per formattazioni multiple, utilizzando dati in sequenza.

EXAMPLE

NOTES

BUGS

SEE ALSO
assembly/asmgraphics.h
exec.library/RawDoFmt()
intuition.library/EasyRequest()
intuition/intuition.h

NAME
TextFmtSizeArgs -- Rettangolo occupato da una stringa formattata.
TextFmtSize -- Varargs stub for TextFmtSizeArgs().

SYNOPSIS
```
TextFmtSizeArgs(rp, ibox, TextFmt, ArgList)
                A1  A3     A0      A2

void TextFmtSizeArgs(struct RastPort *, struct IBox *, STRPTR, APTR);

TextFmtSize(rp, ibox, TextFmt, Arg1, Arg2, ...)

void TextFmtSize(struct RastPort *, struct IBox *, STRPTR, ...);
```

FUNCTION
Determina l'area occuapata dal testo TextFmt, nella RastPort rp.
I dati vengono scritti in una struttura IBox, e descrivono la
grandezza ed altezza in pixel del testo, considerando come
LeftEdge e TopEdge le coordinate (0,0). Le prime due WORD della
struttura IBox, vengono infatti sempre azzerate, basta comunque
sommare a ibox_Width e ibox_Height le vostre coordinate iniziali.

INPUTS
rp - Indirizzo della stessa RastPort dove si stamperà il testo.
ibox - Indirizzo di una struttura IBox.
TextFmt - Indirizzo ad una stringa, in formato c-style.
ArgList - Array degli argomenti per il comandi do formattazione.

RESULT
La struttura IBox, passata negli INPUTS, viene così riempita:

```
ibox_Left   = NULL
ibox_Top    = NULL
ibox_Width  = larghezza del testo, in pixel.
ibox_Height = altezza del testo, in pixel.
```

EXAMPLE

NOTES

BUGS

SEE ALSO
graphics.library/TextExtend(), TextFit(), TextLength()
graphics/text.h graphics/rastport.h
intuition/intuition.h

NAME
UnitInfo -- Prende informazioni da una unità.

SYNOPSIS
```
info = UnitInfo (volumename)
 D0                 A0

struct InfoData *UnitInfo ( STRPTR );
```

FUNCTION
Questo comando restituisce la struttura di tipo InfoData, relativa al nome dell'unità passata in volumename.

INPUTS
volumename - Nome del volume. Es. "DF0:", "DF1:" o "Work:" etc...

RESULT
info - Indirizzo di una struttura InfoData. La stessa che ritorna dopo la chiamata al comando dos.library/Info(). Se NULL errore.

EXAMPLE

NOTES

BUGS

SEE ALSO
dos.library/Info()
exec.library/FreeVec()

NAME
UnlockREI -- Restituisce il controllo ad una REI.

SYNOPSIS
```
rei = UnlockREI (lockrei, name)
D0                A0      A1

struct REI *UnlockREI(struct REI *, STRPTR);
```

FUNCTION
Sblocca una REI precedentemente bloccata dal comando LockREI(). Oltre ad riattivare le funzione della finestra associata alla REI, questo comando respristina il puntatore del Mouse, precedentemente posto nello stato di attesa.
Se entrambi gli INPUTS sono a NULL questa funzione fallisce, a differenza di LockREI() che provvede a prendere il puntatore della REI attualmente attiva; ciò è stato fatto per motivi di sicurezza.

INPUTS
lockrei - Puntatore ad una struttura REI, precedentemente bloccata tramite il comando LockREI(). Se NULL verrà preso in considerazione il nome della REI.
name - Puntatore ad una stringa terminata nulla, che contiene il nome della REI che si desidera sbloccare. Questo parametro viene tenuto in considerazione SOLO quando rei == NULL.

RESULT
rei = Se NULL non è stato possibile sbloccare la REI, o la REI NON era stata precedentemente bloccata. Altrimenti viene restituito l'indirizzo della REI sbloccata con successo.

EXAMPLES

NOTES

BUGS

SEE ALSO
LockREI()

NAME
ValueToStringBin -- Converte un valore in binario.

SYNOPSIS
ValueToStringBin(buffer, value, optlen)
A0 D0:32 D1

```
void ValueToStringBin(APTR, LONG, ULONG);
```

FUNCTION
Converte un valore LONG a 32bit, con segno, in una stringa ASCII, in notazione binaria.
ValueToStringBin() pone di default, 32 caratteri nel buffer, quando optlen è NULL, altrimenti, optlen, decide quanti caratteri saranno posti nel buffer.

INPUTS
buffer - Puntatore ad un buffer che conterrà la stringa.
value - Valore LONG da convertire.
optlen - Numero di caratteri che devono essere introdotti nel buffer finale. Se NULL, il numero di caratteri è 32.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
StringBinToValue()
ValueToStringDec(), ValueToStringHex()

NAME
ValueToStringDec -- Converte un valore LONG in ASCII.

SYNOPSIS
```
ValueToStringDec(buffer, value, optlen)
                 A0    D0:32    D1

void ValueToStringDec(APTR, LONG, ULONG);
```

FUNCTION
Converte un valore LONG a 32bit, con segno, in una stringa ASCII, in notazione decimale.
Se il valore passato negli inputs è negativo, nella stringa finale verrà introdotto il carattere '-'.
ValueToStringDec() ha due modi di effetuare la conversione, a secondo dello stato del parametro optlen, il quale determina la lunghezza della stringa finale, introducendo il carattere '0' quando è necessario.

INPUTS
buffer - Puntatore ad un buffer che conterrà la stringa.
value - Valore LONG da converire.
optlen - Numero di caratteri che devono essere introdotti nel buffer finale. Se NULL, il numero di caratteri è direttamente collegato alla conversione del valore. Vedi EXAMPLES.

RESULT

EXAMPLE
```
**
** Uso del parametro optlen
**

    value=255    optlen=NULL      buffer = "255"
    value=255    optlen=4         buffer = "0255"
    value=255    optlen=5         buffer = "00255"
    value=4      optlen=5         buffer = "00004"

**
** Conversione di numeri negativi
**

    value=5      optlen=10         buffer = "0000000005"
     value=-5     optlen=10         buffer = "-0000000005"
    value=-5     optlen=NULL       buffer = "-5"
```

NOTES

BUGS

SEE ALSO
StringDecToValue()
ValueToStringBin(), ValueToStringHex()

NAME
ValueToStringHex -- Converte un valore in esadecimale.

SYNOPSIS
ValueToStringHex(buffer, value, optlen, prefix)
A0 D0:32 D1 D2

void ValueToStringHex(APTR, LONG, ULONG, BOOL);

FUNCTION
Converte un valore LONG a 32bit, con segno, in una stringa ASCII, in notazione esadecimale.
Se il valore passato negli inputs è negativo, nella stringa finale verrà introdotto il carattere '-'.
ValueToStringHex() pone di default, 8 caratteri nel buffer, quando optlen è NULL, altrimenti, optlen, decide quanti caratteri saranno posti nel buffer.

INPUTS
buffer - Puntatore ad un buffer che conterrà la stringa.
value - Valore LONG da convertire.
optlen - Numero di caratteri che devono essere introdotti nel buffer finale. Se NULL, il numero di caratteri è 8.
prefix - Se TRUE, verrà inserito il carattere '$' nel buffer.

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
StringHexToValue()
ValueToStringDec(), ValueToStringBin()

NAME
WaitREIMsg -- Gestisce tutti gli eventi di una REI.

SYNOPSIS
```
rmsg = WaitREIMsg (rei, underscore)
 D0                A0     D0:8

struct REIMessage *WaitREIMsg (struct REI *, UBYTE);
```

FUNCTION
Usare WaitREIMsg() al posto dell'usuale exec.library/GetMsg() e di gadtools.library/GT_GetIMsg(), per processare gli eventi di una REI. WaitREIMsg() pone il Task in attesa, fino a quando un valido evento non arriva alla porta della Window appartenente alla REI. A questo punto, WaitREIMsg(), farà delle operazioni interne, tali da gestire tutto il sistema REI (Gadget, Window, Menu etc...). Quando viene resrtituito il controllo al Task, significa quindi che un evento si è verificato, e WaitREIMsg(), ci informa di tale evento, tramite la struttura REIMessage:

```
struct REIMessage {
      ULONG rim_Class;
      UWORD rim_Code;
      UWORD rim_Qualifier;
      APTR  rim_IAddress;
      ULONG rim_REICode;
      };
```

Come si può ben vedere, la struttura REIMessage, non è altro che una copia di una parte della struttura IntuiMessage, una copia di quei campi che più ci interessano. Essendo la IntuiMessage di una Window processata dalla gadtools.library, anche i campi della REIMessage, conterranno tutte quelle informazioni aggiuntive, per la gestione dei gadgets della gadtools.

NEW (V41.2) - Per scavalvare ad alcune mancanze di Intuition, il campo REICode ora contiene altre speciali informazioni di aiuto nella gestione degli eventi. E' stata introdotta la gestione del DoubleClick, sia per il tasto sinistro che per il tasto destro del Mouse, questo evento DEVE essere sempre controllato in congiunzione con IDCMP_MOUSEBUTTONS che ci avverte solo della pressione del tasto (in questo caso del sinistro). Il campo Code, tramite SELECTDOWN e MENUDOWN ci informa invece su quale pulsante è stato premuto, rispettivamente sinistro o destro. REICode a sua volta con RIM_LEFTDOUBLECLICK e RIM_RIGHTDOUBLECLICK ci informa se c'è stato un DoubleClick sul tasto sinistro o destro.
Il campo REICode viene anche utilizzato come informazione del Gadget premuto nell'AsmRequest, quando ai pulsante di quest'ultimo vengono agganciati degli Hook. In questo modo l'Hook può sapere quale pulsante dell'AsmRequest è stato premuto.

(GADTOOLS) - Ricordo brevemente che ad esempio il campo Code ci fornisce il numero dell'elemento selezionato in un LIST_VIEW, o di un gadget di tipo CYCLE, etc.

WaitREIMsg() introduce comunque, un modo nuovo di gestire i messaggi che arrivano ad una Window, sicuramente con molti vantaggi rispetto alla procedura standar. Infatti la prima facilitazione che abbiamo è quella di non dover più preoccuparci di rispondere al messaggio, ciò viene fatto automaticamente da WaitREIMsg().
Inoltre con questo sistema, sia i programmatori C che quelli Assembly potranno scegliere come rispondere ai messaggi così ricevuti.

WaitREIMsg() permette sia la gestione nella procedura standar, sia una gestione degli eventi, completamente automatizzata. Per gestione automatizzata, intendiamo l'esecuzione automatica di una routine collegata ad un Gadget o Menu, tramite i comando SetAsmGadgetAttrsA(). SetAsmGadgetAttrsA(), svolge numerose funzioni di inizializzazione, ma non tutte sono obbligatorie. Collegare un AsmHook ad un Gadget, non è necessario per la sua gestione, ne facilita solo l'uso, ma ciò non impedisce di gestiere un messaggio di IDCMP_GADGETUP nella procedura standar. WaitREIMsg() infatti, restituisce sempre una struttura REIMessagge, in qualsiasi caso, informandoci sempre e comunque di quale evento si è verificato.

Oltre alla gestione delle routine collegate agli oggetti, WaitREIMsg() permette di ottenere un controllo maggiore su tutte le operazioni che riguardano i gadgets della GadTools.library.
WaitREIMsg()in pratica è in grado di svolgere automaticamente tutte quelle operazioni che seguono i messaggi di IDCMP_GADGETUP o di IDCMP_GADGETDOWN, permettendo una più rapida gestione dei Gadget. Vediamo qui di seguito, quali funzioni automatiche e non, possono essere svolte, tramite WaitREIMsg().

1) - Se un AsmGadget è stato inizializzato tramite SetAsmGadgetAttrsA(), e contiene quindi un puntatore ad una struttura Hook anch'essa inizializzata, WaitREIMsg() invoca questo Hook, cedendo il controllo alla routine di Hook.
L'Hook degli AsmGadget, viene invocato, seguendo perfettamente le regole generali degli Hook standar Amiga, avremo dunque:

object == (struct AsmGadget *) e message == (struct REIMessage *)

Per i programmatori Assembly:

A0 = Indirizzo della Struttura Hook
A1 = message -> (struct REIMessage *)
A2 = object -> (struct AsmGadget *)
A6 = puntatore all'assemblyBase (struct AssemblyBase *)

Ricordo ancora, che i registri A0,A1,A2,A6 vanno salvati, e che l'Hook deve sempre restituire NULL come codice di ritorno.

WaitREIMsg() oltre ad eseguire l'eventuale Hook associato al Gadget controlla automaticamente le proprietà di un particolare Gadget. In particolare potremmo dire, che svolge quelle operazioni grafiche non gestite dalla gadtools.

KEY/UNDERSCORE CONTROL - Permette di simulare la pressione del Gadget tramite la tastiera. Riconosce automaticamente se quel particolare tasto premuto è collegato a qualche Gadget. Dal sistema operativo 2.04, si è introdotta la possibilità di sottolineare un particolare carattere nel testo di un Gadget.
Tale funzione però era limitata solo ad una rappresentazione grafica e non vi era nessun controllo sul tasto premuto. Era infatti il programmatore che doveva preoccuparsi di gestire gli inputs da tastiera. E' per questo motivo che negli INPUTS di WaitREIMsg() viene richiesto il parametro 'underscore', in quanto (vedi gadtools) il carattere '_' può essere variato, secondo le regole imposte dalla gadtools. Il controllo della tastiera è stato fatto in modo da escludere le combinazioni di tasti particolari. Ad esempio se un Gadget ha stringa '_Mode', per attivare il Gadget si può premere:

'm'
SHIFT + 'm'

```
     CAPS_LOCK + 'm'
```

Queste sono le sole combinazioni consentite, il resto, come ALT, LEFT_AMIGA etc, sono considerate altre combinazioni, per ora riservate per future esansioni.

DISABLE CONTROL - Se si è premuto un tasto, e questo tasto è trovato nel testo di un Gadget, WaitREIMsg() controlla se questo gadget non è disabilitato, in questo caso restituisce il controllo, senza fare nulla.

BUTTON_KIND - Invoca l'Hook, se presente, sia alla pressione di un tasto sia premendo il Gadget con il pulsante sinistro del Mouse. La selezione del Gadget tramite tastiera, viene evidenziata graficamente con la selezione del Gadget stesso, come se la selezione fosse avvenuta tramite Mouse.

STRING_KIND e INTEGER_KIND - Se questo Gadget ha una stringa di testo associata e contiene il carattere '_' o quello quello passato in Underscore (D0) in WaitREIMsg(), viene ATTIVATO automaticamente e il cursore appare su questo Gadget.
Alla pressione dei ENTER e quindi al rilascio del gadget, salta automaticamente all'Hook associato se presente.

CYCLE_KIND - Questo se possiede un testo con sottolineatura viene girato ciclicamente da WaitREIMsg(). E successivamente salta automaticamente all'Hook associato se presente.

In qualsiasi delle precedenti operazioni, la struttura REIMessage viene sempre restituita e opportunamente inizializzata. Ciò vuol dire che se la selezione di un Gadget avviene ad esempio da tastiera il campo rim_IAddress della REIMessage, conterrà l'indirizzo del Gadget 'premuto'.

INPUTS

rei - Puntatore ad una struttura REI, che visualizzi almeno una Window.

underscore - Valore a 8bit che identifica il tipo di Underscore usato per la sottolineature dei Gadget. Questo parametro NON può essere mai NULL, e di default corrisponde al codice 0x5f, ovvero al carattere '_'.

RESULT

rmsg - Indirizzo di una struttura REIMessage.

EXAMPLES

```
/*                  */
/* Wait a standar message */
/*                  */

struct REIMessage *rmsg;

      /* your OpenInterface() and OpenREIA() ... */


while (fine)
{
    if (rmsg = WaitREIMsg(mainrei,0x5F))
    {
        switch(rmsg->rim_Class)
      {
          case IDCMP_MENUPICK:
            /* other Switch and Case evalutation */
```

```
              break;

            case IDCMP_GADGETUP:
              /* other Switch and Case evalutation */

            case IDCMP_CLOSEWINDOW:
              fine = FALSE;
              break;
        }
      }
   }
```

NOTES

Come si vede dall'EXAMPLES, la gestione dei messaggi è stata qui resa estremamente semplice. Non è più necessario infatti rispondere al messaggio, pensarà a tutto WaitREIMsg(), mantenendo anche le code di eventuali messaggi 'ritardatari'.

BUGS

SEE ALSO

SetAsmGadgetAttrsA(), assembly/asmintuition.h
exec.library/GetMsg(), WaitPort()
gadtools.library/GT_GetIMsg(), GT_ReplyIMsg()
intuition/intuition.h