EC4OS3_fr

COLLABORATORS			
	TITLE : EC4OS3_fr		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 13, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

EC4OS3_fr

1.1 Programmer l'AmigaOS avec Amiga-E

Programmer l'AmigaOS avec Amiga-E

****

1.2 Les avantages de l'Amiga-E

Les avantages de l'Amiga-E

****

L'Amiga-E, déjà très populaire auprès des amigaphiles, est dans une période de renaissance avec le compilateur ECX de Leif Salomonsson. ECX est totalement compatible avec EC v3.3a et peut générer aussi bien du code 68020 pour les Amiga classiques que du code PowerPC pour MorphOS. Une version AmigaOne n'est pas prévue mais n'est pas exclue non plus.

Aucun autre langage de programmation ne rend la programmation système plus facile que l'Amiga-E. Parmi les fonctionnalités qui facilitent la vie on peut citer:

* une syntaxe claire et bien structurée

* des modules précompilables

* les types de bas niveau

* les exceptions légères (elles ne sont pas des objects)

* les listes immédiates

* l'assembleur en ligne

Le résultat c'est un style plus fluide, la programmation système est plus sûre et plus responsable qu'il n'est possible avec d'autres langages sur Amiga.

Normalement un bon style et une relecture soigneuse du code doivent permettre d'éviter le recours à Enforcer, MungWall et autres utilitaires de déboguage système. Il est préférable, autant que possible, de prévenir plutôt que de guérir les bogues de programmation système. Les outils de déboguage avancé peuvent avoir des pouvoirs qui semblent magiques, il n'en reste pas moins qu'ils opèrent au niveau machine, c'est-à-dire à un niveau bien plus bas qu'il n'est souhaitable. Il n'y a pas de meilleure arme contre les bogues qu'une bonne lisibilité du code source. Tout ce qui opère à un niveau plus bas que le code source est de trop bas niveau.

À noter si vous voulez copier du code à partir de ce document: gardez à l'esprit qu'un double charactère "\\" dans un amigaguide équivaut en fait à un unique charactère "\".

Pour plus d'informations sur les functions, référez-vous aux Autodocs.

Pour plus d'informations sur les structures et les étiquettes, référez-vous aux Includes.

La documentation indispensable c'est les Autodocs de Commodore, et c'est encore mieux dans le format AmigaGuide : aminet/dev/misc/AmigaOS_guides.lha

Le compilateur EC v3.3a de Wouter van Oortmerssen (version complète) : wouter.fov120.com/e/

Le compilateur ECX de Leif Salomonsson (version démo). : home.swipnet.se/blubbe/ECX

L'Amiga-E mailing list : www.freelists.org/list/positron

1.3 AmigaOS: La gestion des ressources

La gestion des ressources

****

Comme les ressources système sont partagées entre tous les programmes, une stratégie est nécessaire pour garantir une distribution optimale de ces ressources, qui sont toujours trop limitées. La stratégie de l'AmigaOS est rapide et simpliste : le premier demandeur est le premier servi.

Cette stratégie garantie toujours une vitesse maximale, mais requière de la discipline de la part du programmeur pour assurer une distribution optimale :

* les ressources doivent être allouées le plus tard possible (au dernier moment)

* les ressources doivent être libérées le plus tôt possible

* les allocations et les libérations doivent être "parenthésées"

Les prochains chapitres expliquent ce que "parenthésées" veut dire et en quoi les exceptions de l'Amiga-E simplifient grandement les choses.

Il faut aussi garder à l'esprit que :

* toute demande de ressource peut réussir ou échouer

* le programme doit supporter correctement les deux cas

* il faut informer l'utilisateur de l'origine en cas d'erreur

1.4 AmigaOS: Les librairies système

Les librairies système

****

Comme vous le savez déjà, les fonctionnalités système résident principalement dans les librairies. Certaines sont en ROM, la plupart sont dans le répertoire Libs: et d'autres (comme Reqtools et MUI) ne sont pas fournies par Commodore mais font néanmoins partie de la pratique de l'Amiga et sont des outils incontournables pour le programmeur.

Les librairies sont donc les ressources de base et la première chose à savoir c'est comment bien les gérer :

Autant placer OPT OSVERSION=37 au début de chaque programme, il n'y a aucun intérêt à programmer en dessous de l'OS37.

Une librairie s'ouvre avec OpenLibrary(name,version).

Spécifez version=37 pour l'OS2.0+ et version=39 pour l'OS3.0+.

Retenez le résultat de l'appel dans la variable librarybase.

Si le résultat est NIL il n'y a rien d'autre à faire qu'afficher un message d'erreur.

Sinon vous pouvez appeler toutes les fonctions de cette librairie.

Une fois que vous en avez terminé vous devez appeler CloseLibrary(librarybase).

Ensuite vous ne pouvez plus appeler aucune fonction de cette librairie.

La reqtools.library est bien pratique pour afficher des messages d'erreur, c'est pourquoi il est très commode de l'ouvrir avant toute autre chose. Le programme HelloWorld suivant illustre l'ouverture d'une librairie :

```
OPT OSVERSION=37
MODULE 'reqtools'
PROC main()
reqtoolsbase:=OpenLibrary('reqtools.library',37)
IF reqtoolsbase
RtEZRequestA('Hello World!','Ok',0,0,0)
CloseLibrary(reqtoolsbase)
ELSE
PrintF('Could not open reqtools.library!\n')
ENDIF
ENDPROC
```

La variable de base est "reqtoolsbase" et "RtEZRequestA" affiche un simple panneau.

Notez que le succès et l'échec sont deux cas distincts et clairement séparés :

En cas d'échec d'OpenLibrary : on affiche un message d'erreur (et on ne peut pas utiliser reqtools pour le faire !), on ne ferme pas la librairie.

En cas de succès d'OpenLibrary : on peut appeler les fonctions reqtools, et on doit fermer la librairie quand on en a terminé.

C'est ça la gestion "parenthésée" des ressources.

1.5 Amiga-E: Ressources et exceptions Amiga-E

Ressources et exceptions Amiga-E (resource tracking)

****

Le programme précédent est limpide mais pas très réaliste. Un programme réaliste comporte bien davantage de librairies, de fichiers et autres allocations mémoire. Et cette profusion de ressources ne doit pas se faire au détriment de la clarté du code source. On ne peut pas imbriquer des IF indéfiniment, ou alors même le programme le plus simple paraîtra abominablement tortueux. Heureusement, les exceptions Amiga-E permettent de rendre invisible la structuration due à la gestion parenthésée des ressources. La gestion des ressources devient encore plus sûre, et la logique du programme encore plus transparente.

Voici à quoi ressemble le HelloWorld modifié pour les exceptions :

```
OPT OSVERSION=37

MODULE 'reqtools'

ENUM ERR_OK,ERR_NOREQTOOLS

PROC main() HANDLE

reqtoolsbase:=OpenLibrary('reqtools.library',37)

IF reqtoolsbase=NIL THEN Raise(ERR_NOREQTOOLS)

RtEZRequestA('Hello World!','Ok',0,0,0)

EXCEPT DO

SELECT exception

CASE ERR_NOREQTOOLS

PrintF('Could not open reqtools.library v37+ !\n')

ENDSELECT

IF reqtoolsbase THEN CloseLibrary(reqtoolsbase)

ENDPROC
```

Cette version peut paraître plus compliquée que la précédente, elle a néanmoins un avantage décisif : ajouter une ressource ne modifie pas la structure du code, ça n'est plus que l'affaire d'un "Open", d'un "Raise", d'un "CASE" et d'un "Close" supplémentaire.

Toutefois n'utilisez pas les exceptions automatiques : elles rendent les choses implicites, ce qui n'est généralement pas une bonne idée. Rendez les choses aussi explicites que possible.

1.6 Amiga-E: Programme et environnement

Programme et environnement

****

Un programme peut être lancé soit à partir du Shell soit à partir du Workbench. Les deux cas doivent être gérés de façon appropriée. La variable wbmessage permet de connaître l'environnement de lancement. Si le programme est lancé à partir du Workbench alors on a wbmessage<>NIL. Si le programme est lancé à partir du Shell alors on a wbmessage=NIL.

Bien sûr, si le programme est lancé à partir du Shell alors on préfère un message dans la console plutôt qu'un panneau. Il faut alors utiliser wbmessage pour distinguer les deux cas.

Une autre attente du Shell c'est le code d'erreur retourné :

* 0 si tout est OK.

* ou 5 pour un avertissement (le programme peut tout de même poursuivre).

* ou 10 pour une erreur (le programme ne peut pas poursuivre).

Et voici à quoi ressemble le HelloWorld modifié pour exploiter au mieux l'environnement de démarrage :

```
OPT OSVERSION=37
MODULE 'reqtools'
ENUM ERR_OK,ERR_NOREQTOOLS
PROC main() HANDLE
DEF rcode
reqtoolsbase:=OpenLibrary('reqtools.library',37)
IF reqtoolsbase=NIL THEN Raise(ERR_NOREQTOOLS)
display('Hello World!')
EXCEPT DO
SELECT exception
CASE ERR_NOREQTOOLS
PrintF('Could not open reqtools.library v37+ !\n')
rcode:=10
DEFAULT
rcode:=0
ENDSELECT
IF reqtoolsbase THEN CloseLibrary(reqtoolsbase)
ENDPROC rcode
PROC display(msg)
IF wbmessage
RtEZRequestA(msg,'Ok',0,0,0)
ELSE
PrintF('\s\n',msg)
ENDIF
ENDPROC
```

1.7 AmigaOS: Les arguments Shell

Les arguments Shell

****

Les paramètres DOS sont interprétés par la fonction ReadArgs() de la dos.library, on fournit un tableau de LONGs à ReadArgs(), et en retour ReadArgs() y stocke toutes les valeurs de paramètres DOS.

Bien entendu il faut appeler FreeArgs() quand on a terminé.

Cet exemple simple affiche le MESSAGE fournit comme argument shell :

```
OPT OSVERSION=37
ENUM ERR_OK,ERR_ARGS
DEF myargs:PTR TO LONG,rdargs
```

```
PROC main() HANDLE
myargs:=[0]
rdargs:=ReadArgs('MESSAGE/A',myargs,NIL)
IF rdargs=NIL THEN Raise(ERR_ARGS)
PrintF('\s\n',myargs[0])
EXCEPT DO
SELECT exception
CASE ERR_ARGS
PrintF('Bad Args!\n')
ENDSELECT
IF rdargs THEN FreeArgs(rdargs)
ENDPROC
```

Pour plus d'information sur ReadArgs() et les paramères DOS, voir les Autodocs.

1.8 AmigaOS: Les arguments Workbench

Les arguments Workbench

****

Les types d'outil (tooltypes) sont vraiment pratiques, et même plus pratiques que les paramètres DOS car ils sont persistants. Donc les types d'outil sont utiles même si on démarre du Shell. Ils fournissent les paramètres de base que l'utilisateur Shell pourra affiner par des paramètres DOS.

Voici comment faire pour lire les types d'outil :

1. ouvrir la icon.library

Puis si on démarre du Shell:

2. obtenir le chemin d'accès du programme,

via GetProgramName() de la dos.library

Ou si on démarre du Workbench:

2a. obtenir le chemin du répertoire wbmessage.arglist.lock,

via NameFromLock() de la dos library

2b. ajouter le nom du programme wbmessage.arglist.name,

via AddPart() de la dos library

Continuer:

3. charger l'objet icône du programme via GetDiskObject() de la icon.library

4. lire les types d'outil via FindToolType() de la icon.library,

les types d'outils inexistants retournent NIL

5. une fois terminé, libérer l'objet icône via FreeDiskObject()

Le code suivant illustre l'ensemble du processus, il lit le type d'outil MESSAGE dans le fichier .info du programme, et l'affiche dans le Shell ou sur le Workbench :

```
OPT OSVERSION=37
MODULE 'reqtools','icon'
```

```
MODULE 'workbench/startup','workbench/workbench'
ENUM ERR_OK,ERR_NOREQTOOLS,ERR_NOICON,ERR_NOINFO,ERR_NOMESSAGE
DEF prog[80]:STRING
DEF wbmsg:PTR TO wbstartup
DEF diskobj:PTR TO diskobject
PROC main() HANDLE
DEF rcode,msg
reqtoolsbase:=OpenLibrary('reqtools.library',37)
IF reqtoolsbase=NIL THEN Raise(ERR_NOREQTOOLS)
iconbase:=OpenLibrary('icon.library',37)
IF iconbase=NIL THEN Raise(ERR_NOICON)
IF wbmessage
wbmsg:=wbmessage
NameFromLock(wbmsg.arglist.lock,prog,80)
AddPart(prog,wbmsg.arglist.name,80)
ELSE
GetProgramName(prog,80)
ENDIF
IF (diskobj:=GetDiskObject(prog))=NIL THEN Raise(ERR_NOINFO)
msg:=FindToolType(diskobj.tooltypes,'MESSAGE')
IF msg=NIL THEN Raise(ERR_NOMESSAGE)
display(msg)
EXCEPT DO
rcode:=10
SELECT exception
CASE ERR_NOREQTOOLS
PrintF('Could not open reqtools.library v37+ !\n')
CASE ERR_NOICON
display('Could not open icon.library v37+ !')
CASE ERR_NOINFO
display('Could not open .info file!')
CASE ERR_NOMESSAGE
display('Could not find MESSAGE tooltype!')
DEFAULT
rcode:=0
ENDSELECT
IF diskobj THEN FreeDiskObject(diskobj)
IF iconbase THEN CloseLibrary(iconbase)
IF reqtoolsbase THEN CloseLibrary(reqtoolsbase)
```

```
ENDPROC rcode
PROC display(msg)
IF wbmessage
RtEZRequestA(msg,'Ok',0,0,0)
ELSE
PrintF('\s\n',msg)
ENDIF
ENDPROC
```

1.9 AmigaOS: Le versionnement

Le versionnement

****

Une chaîne de version de programme est identique à n'importe quelle autre chaîne de version :

```
OPT OSVERSION=37
PROC main()
PrintF('Hello World!\n')
ENDPROC
CHAR '$VER: HelloWorld 1.0 (04.05.2004) Copyright © Damien Guichard'
```

La commande C:Version permet d'afficher cette information de version.

Si vous ne l'avez pas déjà, préférez plutôt VersionWB de Håkan Parting :

aminet/util/sys/VersionWB.lha

1.10 AmigaOS: La localisation

La localisation

****

C'est le domaine de la locale.library.

Localiser un programme n'est vraiment pas compliqué.

Un catalogue contient toutes les chaînes du programme.

Il est ouvert par OpenCatalogA() et fermé par CloseCatalog().

Entre-temps toutes les chaînes sont lues via GetCatalogStr().

Utilisez EasyCatalog pour produire le catalogue :

aminet/dev/misc/EasyCatalog.lha

Et voici comment localiser un HelloWorld :

```
OPT OSVERSION=37
MODULE 'locale'
ENUM ERR_OK,ERR_NOLOCALE
ENUM CAT_HELLO
```

```
DEF cat
PROC main() HANDLE
localebase:=OpenLibrary('locale.library',38)
IF localebase=NIL THEN Raise(ERR_NOLOCALE)
cat:=OpenCatalogA(NIL,'hello.catalog',NIL)
PrintF(GetCatalogStr(cat,CAT_HELLO,'Hello World!\n'))
EXCEPT DO
SELECT exception
CASE ERR_NOLOCALE
PrintF('Could not open locale.library v38+ !\n')
ENDSELECT
IF cat THEN CloseCatalog(cat)
IF localebase THEN CloseLibrary(localebase)
ENDPROC
```

Ce programme pourrait écrire 'Bonjour le monde!\n' avec le catalogue approprié et "français" comme langue sélectionnée.

À noter qu'il n'y a pas besoin de quitter si OpenCatalogA échoue.

Dans ce cas le programme utilise simplement les chaînes internes.

Note importante: si "english" est le langage sélectionné alors OpenCatalogA n'ouvrira PAS de catalogue, et ce même si un catalogue "english" est présent. Dans ce cas OpenCatalogA renvoie toujours NIL, c'est pourquoi il faut sélectionner "français" pour tester la localisation.

1.11 AmigaOS: Les signaux de tâche

Les signaux de tâche (task signals)

****

Les applications à interface graphique sont dirigées par les évènements,

elles sont en repos en attente d'un signal qui les réactive. Une tâche en repos ne consomme aucune puissance processeur. Vous pouvez mettre votre programme en repos en appelant Wait() avec les évènements dont vous voulez être averti.

Par exemple ce programme est en repos jusqu'à réception d'un "Break" émis en appuyant sur le bouton "Break" de PriMan v2.0 :

```
OPT OSVERSION=37
CONST SIGNAL_BREAK=$1000
PROC main()
PrintF('sleeping... (use PriMan v2.0 to continue)\n')
Wait(SIGNAL_BREAK)
PrintF('running...\n')
ENDPROC
```

Téléchargez PriMan v2.0 ici :

aminet/util/moni/Priman20.lha

Plus tard nous verrons comment Wait() peut attendre d'autres évènements comme les évènements de fenêtre.

1.12 Amiga-E: Les listes d'attributs

Les listes d'attributs (tag lists)

****

Les listes d'attributs sont un moyen très flexible pour passer des paramètres aux fonctions AmigaOS. Une liste d'attributs est simplement un tableau de LONGs qui a toujours la forme suivante :

[TAG1,info1,TAG2,info2,...,0]

Une étiquette en majuscules indique quel attribut est renseigné par le LONG "info" suivant.

Le "info" est la valeur de l'attribut.

Le type liste d'attributs est natif en Amiga-E.

Notez que les listes d'attributs sont allouées statiquement.

Pour des listes d'attributs allouées dynamiquement il faut utiliser NEW :

NEW [TAG1,info1,TAG2,info2,...,0]

Parmi de nombreuses autres librairies, Intuition et Gadtools font un usage intensif des listes d'attributs.

Pour plus d'information sur les listes d'attributs, voir les Includes et les Autodocs de la utility.library

1.13 AmigaOS: Les listes Exec

Les listes Exec

****

De nombreux éléments de l'AmigaOS, comme par exemple les écrans, sont reliés entre eux par une liste Exec.

Alors qu'une liste d'attributs ressemble fort à un tableau statique, une liste Exec est dynamique comme une liste chaînée, on peut y insérer ou y retirer des éléments.

Les fonctions de base pour ce faire sont AddHead(), AddTail(), RemHead(), RemTail(), Insert() et Remove() de la librairie exec.

Cette fonction alloue une liste Exec vide:

```
MODULE 'exec/nodes','exec/lists'

PROC newlist()

DEF list:PTR TO mlh

list:=NewR(SIZEOF mlh)

list.head:=list+4

list.tailpred:=list

ENDPROC list
```

Et celle-ci retourne un nouveau noeud de liste portant le nom spécifié:

```
PROC newnode(name)

ENDPROC NEW [0,0,0,0,name]:ln
```

Pour plus d'information sur les listes Exec, voir les Autodocs de la exec.library ainsi que les Includes exec/nodes.h et exec/lists.h

1.14 AmigaOS avec Amiga-E: Intuition

Intuition

****

Créer, gérer et fermer une fenêtre intuition, se fait toujours selon le même procédé général:

* fournir les WA_FLAGS, qui spécifient l'équipement de la fenêtre

* fournir les WA_IDCMP, qui spécifient quels évènements sont notifiés

* ouvrir la fenêtre via OpenWindowTagList()

* le port de message de la fenêtre est win.userport

* le signal de tâche correspondant est Shl(1,win.userport.sigbit)

* attendre les signaux voulus, en plus du signal de port de fenêtre

* lors de la réception d'un message par le port de fenêtre, il faut:

* 1. obtenir le message par imsg:=GetMsg(win.userport)

* 2. l'évènement de fenêtre est imsg.class

* 3. éventuellement, le gadget concerné est imsg.iaddress

* 4. répondre au message par ReplyMsg(imsg)

* 5. lors d'un évènement IDCMP_REFRESHWINDOW il faut appeller BeginRefresh(),

* puis redessiner le contenu de la fenêtre, et ensuite appeller EndRefresh()

* avant de fermer la fenêtre, il faut vider le port de message

```
OPT OSVERSION=37
MODULE 'exec/ports'
MODULE 'reqtools','intuition/intuition'
ENUM ERR_OK,ERR_NOREQTOOLS,ERR_NOWINDOW
CONST SIGNAL_BREAK=$1000
DEF win:PTR TO window
DEF iclass,icode,igad:PTR TO gadget
PROC main() HANDLE
reqtoolsbase:=OpenLibrary('reqtools.library',37)
IF reqtoolsbase=NIL THEN Raise(ERR_NOREQTOOLS)
win:=OpenWindowTagList(NIL,
[WA_TITLE, 'My Window',
WA_FLAGS, WFLG_SIZEGADGET+WFLG_DEPTHGADGET+WFLG_CLOSEGADGET+
WFLG_DRAGBAR+WFLG_SIMPLE_REFRESH+WFLG_ACTIVATE,
WA_IDCMP, IDCMP_CLOSEWINDOW+IDCMP_REFRESHWINDOW,
WA_MINWIDTH, 140,
WA_MINHEIGHT, 35,
NIL])
IF win=NIL THEN Raise(ERR_NOWINDOW)
handle()
```

```
EXCEPT DO
SELECT exception
CASE ERR_NOREQTOOLS
PrintF('Could not open reqtools.library v37+ !\n')
CASE ERR_NOWINDOW
RtEZRequestA('Could not open window!','Ok',0,0,0)
ENDSELECT
IF win THEN CloseWindow(win)
IF reqtoolsbase THEN CloseLibrary(reqtoolsbase)
ENDPROC
PROC handle() HANDLE
DEF signals,winsig,imsg:PTR TO intuimessage
winsig:=Shl(1,win.userport.sigbit)
LOOP
signals:=Wait(winsig+SIGNAL_BREAK)
IF signals AND winsig
WHILE imsg:=GetMsg(win.userport)
iclass:=imsg.class
icode:=imsg.code
igad:=imsg.iaddress
ReplyMsg(imsg)
SELECT iclass
CASE IDCMP_REFRESHWINDOW
BeginRefresh(win)
EndRefresh(win,TRUE)
CASE IDCMP_CLOSEWINDOW
Raise(0)
ENDSELECT
ENDWHILE
ENDIF
IF signals AND SIGNAL_BREAK
Raise(0)
ENDIF
ENDLOOP
EXCEPT
WHILE imsg:=GetMsg(win.userport)
ReplyMsg(imsg)
ENDWHILE
ENDPROC
```

Notez qu'il n'est pas besoin d'ouvrir/fermer la intuition.library, c'est l'AmigaE qui s'en charge.

Pour plus d'information sur les flags WA_FLAGS,WA_IDCMP, sur OpenWindowTagList(),intuimessage, BeginRefresh() et EndRefresh(), voir les Autodocs de la intuition.library ainsi que les Includes intuition/intuition.h

1.15 AmigaOS avec Amiga-E: Gadtools

Gadtools

****

Si nous voulons des gadgets tout prêts à utiliser et des menus faciles à créer, alors intuition n'est pas suffisant, il faut en plus un kit de construction de GUI.

Gadtools est un bon candidat pour démarrer, plus tard vous choisirez BOOPSI, MUI, Reaction ou Feelin, mais d'abord il faut apprendre les techniques de base avec Gadtools.

Gadtools fonctionne à peu près comme intuition, les différences sont:

* vous devez ouvrir/fermer la gadtools.library

* GetMsg/ReplyMsg deviennent Gt_GetIMsg/Gt_ReplyIMsg

* BeginRefresh/EndRefresh deviennent Gt_BeginRefresh/Gt_EndRefresh

* il y a une initialisation en plus à l'aide de GetVisualInfoA() et CreateContext()

* une fois la fenêtre ouverte il faut appeller Gt_RefreshWindow()

Comme nous voulons aussi profiter des facilités de Gatools en matière de menus:

* on crée un menu à l'aide de CreateMenuA() et LayoutMenuA()

* on libère ce menu avec ClearMenuStrip() et FreeMenus()

C'est quand même du code plutôt rébarbatif, c'est pourquoi il est habile de le placer dans un module 'gt_window', ensuite il n'y aura plus qu'à le paramétrer avec les options voulues:

```
OPT MODULE
OPT EXPORT
MODULE 'exec/ports'
MODULE 'reqtools','intuition/intuition'
MODULE 'gadtools','libraries/gadtools'
ENUM ERR_OK,ERR_NOREQTOOLS,ERR_NOGADTOOLS,ERR_NOSCREEN,
ERR_NOVISUAL,ERR_NOCONTEXT,ERR_NOMENU,ERR_NOGADGET,ERR_NOWINDOW
CONST SIGNAL_BREAK=$1000
PROC gt_openwindow(windef,newmenu,winhandler) HANDLE
DEF win=NIL:PTR TO window,iclass,icode,igad
DEF signals,winsig,imsg:PTR TO intuimessage
DEF scr=NIL,visual=NIL,context,gadlist=NIL,menu=NIL
reqtoolsbase:=OpenLibrary('reqtools.library',37)
IF reqtoolsbase=NIL THEN Raise(ERR_NOREQTOOLS)
gadtoolsbase:=OpenLibrary('gadtools.library',37)
IF gadtoolsbase=NIL THEN Raise(ERR_NOGADTOOLS)
IF (scr:=LockPubScreen('Workbench'))=NIL THEN Raise(ERR_NOSCREEN)
IF (visual:=GetVisualInfoA(scr,NIL))=NIL THEN Raise(ERR_NOVISUAL)
IF (context:=CreateContext({gadlist}))=NIL THEN Raise(ERR_NOCONTEXT)
IF newmenu
IF (menu:=CreateMenusA(newmenu,NIL))=NIL THEN Raise(ERR_NOMENU)
```

```
IF LayoutMenusA(menu,visual,NIL)=FALSE THEN Raise(ERR_NOMENU)
ENDIF
IF (win:=windef(scr,context,visual,gadlist))=NIL THEN Raise(ERR_NOWINDOW)
IF menu THEN IF SetMenuStrip(win,menu)=FALSE THEN Raise(ERR_NOMENU)
Gt_RefreshWindow(win,NIL)
winsig:=Shl(1,win.userport.sigbit)
LOOP
signals:=Wait(winsig+SIGNAL_BREAK)
IF signals AND winsig
WHILE imsg:=Gt_GetIMsg(win.userport)
iclass:=imsg.class
icode:=imsg.code
igad:=imsg.iaddress
Gt_ReplyIMsg(imsg)
winhandler(win,iclass,icode,igad)
ENDWHILE
ENDIF
IF signals AND SIGNAL_BREAK
Raise(0)
ENDIF
ENDLOOP
EXCEPT DO
WHILE imsg:=Gt_GetIMsg(win.userport)
Gt_ReplyIMsg(imsg)
ENDWHILE
SELECT exception
CASE ERR_NOREQTOOLS
PrintF('Could not open reqtools.library v37+ !\n')
CASE ERR_NOGADTOOLS
RtEZRequestA('Could not open gadtools.library v37+ !','Ok',0,0,0)
CASE ERR_NOSCREEN
RtEZRequestA('Could not lock Workbench screen!','Ok',0,0,0)
CASE ERR_NOVISUAL
RtEZRequestA('Could not get visual info!','Ok',0,0,0)
CASE ERR_NOCONTEXT
RtEZRequestA('Could not create window context!','Ok',0,0,0)
CASE ERR_NOMENU
RtEZRequestA('Could not create menu!','Ok',0,0,0)
CASE ERR_NOGADGET
```

```
RtEZRequestA('Could not create gadget!','Ok',0,0,0)
CASE ERR_NOWINDOW
RtEZRequestA('Could not open window!','Ok',0,0,0)
ENDSELECT
IF win THEN ClearMenuStrip(win)
IF menu THEN FreeMenus(menu)
IF win THEN CloseWindow(win)
IF gadlist THEN FreeGadgets(gadlist)
IF visual THEN FreeVisualInfo(visual)
IF scr THEN UnlockPubScreen(NIL,scr)
IF gadtoolsbase THEN CloseLibrary(gadtoolsbase)
IF reqtoolsbase THEN CloseLibrary(reqtoolsbase)
ENDPROC
```

Voilà, maintenant nous pouvons ouvrir une fenêtre en appelant la fonction gt_openwindow() avec la définition de fenêtre et le gestionnaire appropriés:

```
OPT OSVERSION=37
MODULE 'intuition/intuition','gadtools'
MODULE '*gt_window'
PROC main()
gt_openwindow({windef},NIL,{winhandler})
ENDPROC
PROC windef(scr,context,visual,gadlist)
RETURN OpenWindowTagList(NIL,
[WA_TITLE, 'My Window',
WA_FLAGS, WFLG_SIZEGADGET+WFLG_DEPTHGADGET+WFLG_CLOSEGADGET+
WFLG_DRAGBAR+WFLG_SIMPLE_REFRESH+WFLG_ACTIVATE,
WA_IDCMP, IDCMP_CLOSEWINDOW+IDCMP_REFRESHWINDOW,
WA_MINWIDTH, 140,
WA_MINHEIGHT, 35,
WA_GADGETS, gadlist,
NIL])
ENDPROC
PROC winhandler(win,iclass,icode,igad)
SELECT iclass
CASE IDCMP_REFRESHWINDOW
Gt_BeginRefresh(win)
Gt_EndRefresh(win,TRUE)
CASE IDCMP_CLOSEWINDOW
Raise(0)
ENDSELECT
ENDPROC
```

1.16 AmigaOS: Les menus Gadtools

Les menus Gadtools

****

En premier il faut ajouter IDCMP_MENUPICK dans les WA_IDCMP de votre fenêtre.

Ensuite il faut fournir une structure newmenu complète.

Alors, chaque fois d'une entrée de menu est sélectionnée, le gestionnaire de fenêtre est appellé avec iclass qui vaut IDCMP_MENUPICK et icode qui désigne le numéro de menu sélectionné.

A partir de ce numéro de menu les 2 fonctions suivantes extraient le numéro de l'article et le numéro du sous-article:

PROC itemnum(n) // menu item

ENDPROC Shr(n,5) AND $3F

PROC subnum(n) // menu sub item

ENDPROC Shr(n,11) AND $1F

Voyez les Includes libraries/gadtools.h et intuition/intuition.h pour plus d'information sur les menus Gadtools.

1.17 AmigaOS: Les gadgets Gadtools

Les gadgets Gadtools

****

La fonction CreateGadgetA() permet de créer et de lier un nouveau gadget Gadtools à une fenêtre.

CreateGadgetA() attend comme arguments un type de gadget tel que SCROLLER_KIND ou LISTVIEW_KIND, ainsi qu'une structure newgadget initialisée et une liste d'attributs.

La structure newgadget contient essentiellement les dimensions du gadget et le visual info.

La liste d'attributs est spécifique au type de gadget, voyez les Includes libraries/gadtools.h pour davantage d'information.

Vous n'avez pas à tester toutes les créations de gadgets, il suffit de tester la dernière création, car elle a échoué si l'une des précédentes a échoué.

Un gadget LISTVIEW_KIND peut recevoir une liste Exec comme contenu initial. Dans ce cas il nous incombe de créer la liste Exec avant la création du gadget LISTVIEW_KIND.

Autre chose, les coordonnées de gadgets sont données relativement au bord extérieur de la fenêtre, et non pas relativement au bord intérieur, c'est pourquoi il faut leur ajouter offx et offy tels qu'ils sont calculés.

Ce calendrier factice illustre la création d'une GUI complète, avec menu, à l'aide de Gadtools:

```
OPT OSVERSION=37

MODULE 'intuition/intuition','intuition/screens','intuition/gadgetclass'

MODULE 'gadtools','libraries/gadtools'

MODULE 'utility/tagitem','graphics/rastport'

MODULE 'exec/nodes','exec/lists'

MODULE '*gt_window'

PROC main()

DEF menu

menu:=[NM_TITLE,0,'Project',0,0,0,0,

NM_ITEM,0,'New','N',0,0,0,
```

```
NM_ITEM,0,'Load...','L',0,0,0,
NM_ITEM,0,'Save','S',0,0,0,
NM_ITEM,0,'Babbling',0,0,0,0,
NM_SUB,0,'aaargh','A',0,0,0,
NM_SUB,0,'hmmm','H',0,0,0,
NM_ITEM,0,NM_BARLABEL,0,0,0,0,
NM_ITEM,0,'Quit','Q',0,0,0,
0,0,0,0,0,0,0]:newmenu
gt_openwindow({windef},menu,{winhandler})
ENDPROC
PROC newlist()
DEF list:PTR TO mlh
list:=NewR(SIZEOF mlh)
list.head:=list+4
list.tailpred:=list
ENDPROC list
PROC insert(list,name)
AddTail(list,NEW [0,0,0,0,name]:ln)
ENDPROC
PROC windef(scr:PTR TO screen,gad,visual,glist)
DEF offx,offy,labels=NIL
offx:=scr.wborleft
offy:=scr.wbortop+scr.rastport.txheight+1
gad:=CreateGadgetA(CYCLE_KIND,gad,
[offx+20,offy+9,155,16,NIL,scr.font,0,0,visual,0]:newgadget,
[GTCY_LABELS,
['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday',
NIL],
TAG_DONE])
gad:=CreateGadgetA(SLIDER_KIND,gad,
[offx+40,offy+36,135,16,NIL,scr.font,0,0,visual,0]:newgadget,
[GTSL_MIN,1,
GTSL_MAX,31,
GTSL_LEVELFORMAT,'\d[2]',
GTSL_MAXLEVELLEN,2,
TAG_DONE])
labels:=newlist()
insert(labels,'January')
insert(labels,'February')
```

```
insert(labels,'March')
insert(labels,'April')
insert(labels,'May')
insert(labels,'June')
insert(labels,'July')
insert(labels,'August')
insert(labels,'September')
insert(labels,'October')
insert(labels,'November')
insert(labels,'December')
gad:=CreateGadgetA(LISTVIEW_KIND,gad,
[offx+20,offy+63,155,100,NIL,scr.font,0,0,visual,0]:newgadget,
[GTLV_SCROLLWIDTH,20,
GTLV_LABELS,labels,
TAG_DONE])
IF gad=NIL THEN Raise(ERR_NOGADGET)
RETURN OpenWindowTagList(NIL,
[WA_TITLE, 'Calendar Demo',
WA_FLAGS, WFLG_DEPTHGADGET+WFLG_CLOSEGADGET+WFLG_DRAGBAR+
WFLG_SIMPLE_REFRESH+WFLG_ACTIVATE,
WA_IDCMP, IDCMP_CLOSEWINDOW+IDCMP_REFRESHWINDOW+
IDCMP_GADGETDOWN+IDCMP_GADGETUP+IDCMP_MOUSEMOVE+IDCMP_MENUPICK,
WA_LEFT, 70,
WA_TOP, 40,
WA_WIDTH, 200,
WA_HEIGHT, offy+176,
WA_GADGETS, glist,
WA_NEWLOOKMENUS, TRUE,
WA_AUTOADJUST, TRUE,
TAG_DONE])
ENDPROC
PROC itemnum(n)
ENDPROC Shr(n,5) AND $3F
PROC winhandler(win,iclass,icode,igad)
SELECT iclass
CASE IDCMP_CLOSEWINDOW
Raise(ERR_OK)
CASE IDCMP_REFRESHWINDOW
Gt_BeginRefresh(win)
```

```
Gt_EndRefresh(win,TRUE)
CASE IDCMP_MENUPICK
IF itemnum(icode)=5 THEN Raise(ERR_OK)
ENDSELECT
ENDPROC
```

Voyez les Includes libraries/gadtools.h pour davantage d'information sur les gadgets Gadtools.

1.18 AmigaOS: Interaction avec gadtools

Interaction

****

L'interaction avec les gadgets Gadtools est similaire à l'interaction avec les menus sauf que class est généralement IDCMP_GADGETUP et que gad désigne le gadget concerné par l'évènement.

Pour aider à l'identification de gad il est pratique de spécifier un gadgetid dans la structure newmenu. Ainsi la réaction à un évènement IDCMP_GADGETUP peut être sélectionnée en fonction de la valeur de gad.gadgetid

La lecture et la mise à jour des attributs de gadgets ne sont pas symétriques.

Par exemple, dans gad.specialinfo.longint on peut lire la value d'un gadget INTEGER_KIND. Par contre pour écrire cette même valeur on doit passer par un appel à la fonction Gt_SetGadgetAttrsA() qui valuera l'étiquette GTIN_NUMBER. C'est ce qui permet à Gadtools d'actualiser l'affichage en plus de modifier la valeur.

Cette calculatrice sommaire illustre l'interaction entre plusieurs gadgets d'une GUI gadtools:

```
OPT OSVERSION=37
MODULE 'intuition/intuition','intuition/screens'
MODULE 'reqtools','gadtools','libraries/gadtools'
MODULE 'utility/tagitem','graphics/rastport'
MODULE '*gt_window'
DEF display:PTR TO gadget
DEF result=0
DEF value=0
PROC main()
DEF newmenu
newmenu:=[NM_TITLE,0,'Calculator',0,0,0,0,
NM_ITEM,0,'Reset','R',0,0,0,
NM_ITEM,0,'About','A',0,0,0,
NM_ITEM,0,'Quit','Q',0,0,0,
0,0,0,0,0,0,0]:newmenu
gt_openwindow({windef},newmenu,{winhandler})
ENDPROC
PROC windef(scr:PTR TO screen,gad,visual,glist)
DEF offx,offy
offx:=scr.wborleft
```

```
offy:=scr.wbortop+scr.rastport.txheight+1
gad:=display:=CreateGadgetA(INTEGER_KIND,gad,
[offx+10,offy+8,124,22,NIL,scr.font,0,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+10,offy+36,40,20,'7',scr.font,7,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+52,offy+36,40,20,'8',scr.font,8,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+94,offy+36,40,20,'9',scr.font,9,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+10,offy+58,40,20,'4',scr.font,4,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+52,offy+58,40,20,'5',scr.font,5,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+94,offy+58,40,20,'6',scr.font,6,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+10,offy+80,40,20,'1',scr.font,1,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+52,offy+80,40,20,'2',scr.font,2,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+94,offy+80,40,20,'3',scr.font,3,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+10,offy+102,40,20,'0',scr.font,0,0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+52,offy+102,40,20,'+',scr.font,"+",0,visual,0]:newgadget,
NIL)
gad:=CreateGadgetA(BUTTON_KIND,gad,
[offx+94,offy+102,40,20,'=',scr.font,"=",0,visual,0]:newgadget,
```

```
NIL)
IF gad=NIL THEN Raise(ERR_NOGADGET)
RETURN OpenWindowTagList(NIL,
[WA_TITLE, 'Minicalc Demo',
WA_FLAGS, WFLG_DEPTHGADGET+WFLG_CLOSEGADGET+WFLG_DRAGBAR+
WFLG_SIMPLE_REFRESH+WFLG_ACTIVATE,
WA_IDCMP, IDCMP_CLOSEWINDOW+IDCMP_REFRESHWINDOW+
IDCMP_GADGETUP+IDCMP_MENUPICK,
WA_LEFT, 70,
WA_TOP, 40,
WA_WIDTH, 150,
WA_HEIGHT, offy+136,
WA_GADGETS, glist,
WA_NEWLOOKMENUS, TRUE,
WA_AUTOADJUST, TRUE,
TAG_DONE])
ENDPROC
PROC itemnum(n)
ENDPROC Shr(n,5) AND $3F
PROC winhandler(win,class,code,gad:PTR TO gadget)
SELECT class
CASE IDCMP_CLOSEWINDOW
Raise(ERR_OK)
CASE IDCMP_REFRESHWINDOW
Gt_BeginRefresh(win)
Gt_EndRefresh(win,TRUE)
CASE IDCMP_GADGETUP
IF gad=display
result:=result+gad.specialinfo::stringinfo.longint
Gt_SetGadgetAttrsA(display,win,NIL,[GTIN_NUMBER,result,0])
value:=0
ELSEIF (gad.gadgetid>=0) AND (gad.gadgetid<=9)
value:=Mul(10,value)+gad.gadgetid
Gt_SetGadgetAttrsA(display,win,NIL,[GTIN_NUMBER,value,0])
ELSEIF gad.gadgetid="+"
result:=result+value; value:=0
Gt_SetGadgetAttrsA(display,win,NIL,[GTIN_NUMBER,0,0])
ELSEIF gad.gadgetid="="
result:=result+value
```

```
Gt_SetGadgetAttrsA(display,win,NIL,[GTIN_NUMBER,result,0])
value:=0; result:=0
ENDIF
CASE IDCMP_MENUPICK
IF itemnum(code)=0
value:=0; result:=0
Gt_SetGadgetAttrsA(display,win,NIL,[GTIN_NUMBER,0,0])
ELSEIF itemnum(code)=1
RtEZRequestA('Demo: Mini Calculator\nAuthor: Damien Guichard\n',
'Ok',0,0,0)
ELSEIF itemnum(code)=2
Raise(ERR_OK)
ENDIF
ENDSELECT
ENDPROC
```

1.19 AmigaOS: Serveurs AREXX

Serveurs AREXX

****

Un serveur AREXX augmente sensiblement l'interopérabilité de votre programme en exportant toutes ses fonctionnalités, ce qui permet de scripter leur utilisation et de coopérer avec une multitude d'autres programmes serveurs AREXX.

L'interaction avec AREXX se fait par la rexxsyslib.library, pour cela vous créer et nommez un port de message, ce port de message deviendra votre adresse de serveur AREXX. Pour créer ce port de message vous utilisez les fonctions CreateMsgPort() et DeleteMsgPort() de la exec.library, mais vous pouvez tout aussi bien réutiliser votre port de fenêtre. Ensuite il suffit de définir la priorité du port de message, puis de le rendre public à l'aide de la fonction AddPort(), à la fin il faudra le retirer à l'aide de la fonction RemPort().

Si vous réutilisez votre port de fenêtre, la fonction IsRexxMsg() vous permettra de distinguer un message AREXX d'un simple message intuition.

Une fois le message rxmsg identifié comme étant un message AREXX:

* rxmsg.args[0] est la ligne de commande

* interprétez cette commande

* définissez le code d'erreur rxmsg.result1 (comme une commande DOS)

* pour retourner une valeur placez la chaîne dans rxmsg.result2

* finalement répondez avec ReplyMsg(rxmsg)

Ces quelques commandes s'inspirent de la graphics.library pour offrir un outil de dessin scriptable par AREXX:

QUIT

MOVE x y

DRAW x y

RECTFILL xmin ymin xmax ymax

ELLIPSE cx cy a b

SETPEN pen

TEXT string

Cette petite source implémente le serveur AREXX pour ces commandes:

```
OPT OSVERSION=37
MODULE 'exec/ports','exec/nodes','intuition/intuition'
MODULE 'rexxsyslib','rexx/storage'
ENUM ERR_OK,ERR_NOREXXSYSLIB,ERR_NOWINDOW
CONST SIGNAL_BREAK=$1000
DEF win:PTR TO window
DEF rastport
DEF xa,ya,xb,yb:LONG
PROC main() HANDLE
rexxsysbase:=OpenLibrary('rexxsyslib.library',36)
IF rexxsysbase=NIL THEN Raise(ERR_NOREXXSYSLIB)
win:=OpenWindowTagList(NIL,
[WA_TITLE, 'AREXX Paint Demo',
WA_FLAGS, WFLG_DEPTHGADGET+WFLG_CLOSEGADGET+WFLG_DRAGBAR+
WFLG_NOCAREREFRESH+WFLG_ACTIVATE,
WA_IDCMP, IDCMP_CLOSEWINDOW,
WA_WIDTH, 320,
WA_HEIGHT, 256,
WA_AUTOADJUST, TRUE,
NIL])
IF win=NIL THEN Raise(ERR_NOWINDOW)
win.userport::ln.name:='APaint.1'
win.userport::ln.pri:=5
AddPort(win.userport)
rastport:=win.rport
SetAPen(rastport,1)
handle()
EXCEPT DO
SELECT exception
CASE ERR_NOREXXSYSLIB
PrintF('Could not open rexxsyslib.library v36+ !\n')
CASE ERR_NOWINDOW
PrintF('Could not open window!\n')
ENDSELECT
IF win THEN CloseWindow(win)
IF rexxsysbase THEN CloseLibrary(rexxsysbase)
```

```
ENDPROC
PROC handle() HANDLE
DEF signals,winsig,iclass,imsg:PTR TO intuimessage
DEF cmdline,rxmsg:PTR TO rexxmsg
winsig:=Shl(1,win.userport.sigbit)
LOOP
signals:=Wait(winsig+SIGNAL_BREAK)
IF signals AND winsig
WHILE rxmsg:=imsg:=GetMsg(win.userport)
IF IsRexxMsg(imsg)
cmdline:=TrimStr(rxmsg.args[0])
IF StrCmp(cmdline,'QUIT')
rxmsg.result1:=0
ReplyMsg(rxmsg)
Raise(0)
ELSEIF StrCmp(cmdline,'MOVE ',STRLEN)
IF read_args(rxmsg,2)
Move(rastport,xa,ya)
ENDIF
ELSEIF StrCmp(cmdline,'DRAW ',STRLEN)
IF read_args(rxmsg,2)
Draw(rastport,xa,ya)
ENDIF
ELSEIF StrCmp(cmdline,'RECTFILL ',STRLEN)
IF read_args(rxmsg,4)
IF (xb>=xa) AND (yb>=ya)
RectFill(rastport,xa,ya,xb,yb)
ELSE
rxmsg.result1:=10
ENDIF
ENDIF
ELSEIF StrCmp(cmdline,'ELLIPSE ',STRLEN)
IF read_args(rxmsg,4)
DrawEllipse(rastport,xa,ya,xb,yb)
ENDIF
ELSEIF StrCmp(cmdline,'SETPEN ',STRLEN)
IF read_args(rxmsg,1)
SetAPen(rastport,xa)
ENDIF
```

```
ELSEIF StrCmp(cmdline,'TEXT ',STRLEN)
cmdline:=TrimStr(cmdline+STRLEN)
Text(rastport,cmdline,StrLen(cmdline))
rxmsg.result1:=0
ENDIF
ReplyMsg(rxmsg)
ELSE
iclass:=imsg.class
ReplyMsg(imsg)
SELECT iclass
CASE IDCMP_CLOSEWINDOW
Raise(0)
ENDSELECT
ENDIF
ENDWHILE
ENDIF
IF signals AND SIGNAL_BREAK
Raise(0)
ENDIF
ENDLOOP
EXCEPT
WHILE imsg:=GetMsg(win.userport)
ReplyMsg(imsg)
ENDWHILE
ENDPROC
PROC read_args(rxmsg:PTR TO rexxmsg,num)
DEF str,ch
str:=TrimStr(rxmsg.args[0])
WHILE str[]<>" " DO INC str
xa,ch:=Val(str); IF ch THEN DEC num; str:=str+ch
ya,ch:=Val(str); IF ch THEN DEC num; str:=str+ch
xb,ch:=Val(str); IF ch THEN DEC num; str:=str+ch
yb,ch:=Val(str); IF ch THEN DEC num; str:=str+ch
IF num THEN rxmsg.result1:=10 ELSE rxmsg.result1:=0
ENDPROC num=0
```

Le Shell permet d'utiliser ces commandes de façon interactive:

```
RUN >NIL: APaint
RX "ADDRESS 'APaint.1' ELLIPSE 160 128 100 60"
RX "ADDRESS 'APaint.1' QUIT"
```

Une chose que ce petit programme ne gère pas ce sont les éxécutions multiples d'un même programme serveur. Dans ce cas vous voulez soit partager un même serveur soit avoir de multiple serveurs, un pour chaque tâche. Pour avoir de multiple serveurs il suffirait de les nommer Apaint.1, APaint.2, APaint.3 et ainsi de suite.

Voyez la rexxsyslib.library ainsi que rexx/storage.h pour plus d'informations sur les serveurs AREXX.

1.20 AmigaOS: Les Gadget-Toolkits

Les Gadget-Toolkits

****

Ce tutoriel ne couvre pas les gadget-toolkits.

Cependant voici où trouver quelques ressources et documentations pour le programmeur Amiga-E:

MUI:

aminet/dev/mui/mui38dev.lha

aminet/dev/mui/mui38dev-E.lha

Feelin:

http://perso.numericable.fr/~olaviale/feelin/