Programming guideline of

version 1.2
October 2002

by
Dominique Béréziat, Arnaud Bercegeay
http://windom.free.fr/

Contents

Appendix

Programming guideline of WinDom

Introduction

Windom is a tool for GEM programming. It allows you to easily handle windows and many other GEM features. This version is now available for Pure C, Sozobon C and Gnu C compiler (with 32 bits int size). WinDom works with all TOS compatible systems (all TOS Atari version, Milan TOS, MiNT, MagiC, MagicMac, MultiTOS, Naes and probably most of PC TOS emulator).

WinDom has two kinds of functions: a new set of GEM functions replacing native GEM functions and other functions realizing complex GEM operations.

Actually, the new GEM functions are very similar to their GEM homolog functions but their actions are extended. For example, using WinDom, we never call appl_init() but the new function ApplInit(). Thus, WinDom programming looks like GEM programming. Some WinDom functions are incompatible with GEM functions but not necessary. The table AES versus Windom in annex lists differences and compatibility between Windom functions and GEM functions.

The concept of WinDom is very simple: each window have a descriptor (a pointer on a C struct). This descriptor contains a set of functions attached to GEM events. The main function of WinDom (the EvntWindom() function) replaces the GEM function evnt_multi(): it intercepts GEM events and execute the right function attached to them. By default, new windows have a set of standard event functions already defined. Then, in most cases, only a small number of event functions has to be written.

Here, a non exhaustiv list of the main features of WinDom:

- Windom tries to unify all AES versions: all Windom functions are available with any AES. When a function uses a special feature (like window iconification) not available on old versions of AES, the feature is emulated.
- Windom has a set of new objects compatible with the famous MyDials library. These objects have a 3D-style available for all TOS, even in monochrom.
- Windom can divide a window in several subwindows (called frames). Each frame is handled like a standard window.
- A function can be attached to an AES message event: when EvntWindom() intercepts this message, the function is executed.

The documentation of Windom is organized as follow: a first part is devoted to a tutorial: each component of WinDom is visited with C code examples, a second part lists all Windom functions grouped by theme. The folder \EXAMPLE of the Windom distribution containts some examples of WinDom programming.

Programming guideline of WinDom

Legal aspects and Contacts

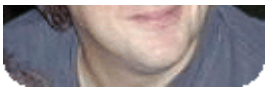
WinDom is a freeware product. It means that it is free of use outside a commercial framework: it cannot be sold nor be included in a commercial distribution with the author's autorization. The author keeps all rights on WinDom (sources files, documentation). You are not allowed to modify the content of the WinDom package whitout the author's authorization. The author cannot be responsable for any inconvenient due to WinDom. If you use WinDom in your program (Public Domain, Freeware, Shareware or Commercial) you have the moral obligation to cite the author of WinDom.

WinDom has been used by the following programs:

- Awele,
- Notes,
- Excellent,
- WinConf,
- Magic Setup,
- Group,
- CD Lab,
- Riri 2,
- Universum,
- GemTIDY,
- ...

WinDom has been written by Dominique Béréziat and Arnaud Bercegeay. For all remarks, critics, comments or suggestion please contact:

Dominique Béréziat
13 rue Georges Sand
91120 PALAISEAU FRANCE
E-mail : Dominique.Bereziat@inria.fr



or

Arnaud Bercegeay
E-mail : Arnaud.Bercegeay@free.fr

New versions of Windom are available at:

http://windom.free.fr

There is a mailing list. Send a message to windom@ml.free.fr with the word 'subscribe' as subject.

Thanks for your attention,
Dominique Béréziat 2001

Programming guideline of WinDom

What's new in this version ?

Programming guideline of WinDom

Version 1.20 (October 2002)

- The FRAMEINFO structure has been hidden from the WINDOW structure (see details below).
- Bag bug fixed in Galloc().
- conf_path() converts correctly absolute unix standard path (e.g. /etc is converted to U:\etc).
- Bag bug fixed in ObjcEdit() : parameter obj is now checked.
- app.work_in had a bad size !! (10 elements instead of 11).
- New prototype for ApplSet() with APS_FLAGS mode (see manual of ApplSet()).
- Color palet handling is disabled in true color mode (e.g. when app.nplanes > 8).
- Bug fixed in color palet updating when a WF_TOP message occures.
- New message WM_PREREDRAW.
- WinDom functions (try to) use standard TOS error specified in the header file toserror.h provided in the package.
- New functions EvntDataAttach(), EvntDataAdd() : user data can be binded to event function. ObjcAttach() and RsrcUserDraw() can bind data too. This new feature is really powerfull and makes the Data library deprecated.
- Due to the previous extension, FormThumb() can now handle severals thumb indexes by formular.
- WinDom can be compiled with the Pure C driver of Julian F. Reschke. It is easier to the author to compile WinDom (a simple gmake command is used to compile WinDom for all supported compiler).
- Extended object XEDIT works better but crashes under Naes.
- Bug fixed in obj_root() and ShelWrite().
- When it is possible (app.nplanes>=4) disabled objects in extended types are displaying with LBLACK color (and not the LIGHT attribut).
- New extended objects XLONGBOXTEXT (a box containing several line of text) and XTEDINFO (as TEDINFO objects but with a same visual aspect however the AES version) (see Extended types and ressource editor section).
- Objects SLIDEPART had been improved.
- Several bugs fixed in frame library and modification :

  - Better aspect of frame widgets (using Arnaud's widgets).
  - New mode APS_FRAMEWIDGETSIZE in ApplSet().
  - Frame structures are now private and hidden (futur enhancement will be more portable from user point of view).
  - New function FrameGet() to access frame informations (now hidden).
  - All examples of frame programming have been recompiled (and adapted to new window specifications).

- app.id contains -1 when AES session is not initialised (i.e. before ApplInit() call).
- New function FormResize().
- New Sliders library to handle slider objects in formulars (see examples/sliders/ demonstration in the distribution).

Programming guideline of WinDom

Version 1.10 (September 2001)

- In this version, many fundamental changes appear. Please, read the section Convert your old WinDom applications if you already have used WinDom,
- Arnaud Bercegeay is now associated at the WinDom Project. Most of new features are supported by him. Its last contribution is the Gcc 32 bits support (thanx to him, it is really a great hacker !).
- WinDom is now available for Gcc ! Read the section Gcc 32 bits portability. Notice it is the main change in this release.
- New functions FormThbGet() and FormThbSet().
- New function EvntRedraw().
- Function ObjcEdit() have new prototype.
- rect_set(), set_clip(), clip_off() respectively renammed in rc_set(), rc_clip_on(), rc_clip_off().
- ExecGemApp() renammed in ShelWrite().
- New mode in RsrcXtype().
- New prototype for ObjcDup() (more object types are supported).
- It is now possible to attach a variable or a function to the desktop menu (see ObjcAttach()).
- New Functions WindTop() and WindFind().
- The undocumented AvServer() function is now officiel and had changed its prototype.
- New WinDom configuration feature: if a variable is not found in the application configuration, it is searched in the default configuration. It is meanning that the default configuration affects all WinDom applications except if addressed variables are defined in the application configuration.
- if the user click on a menu window or a toolbar window without actived an object, the window is topped.
- function rc_intersect() and grect_to_array() had moved in PCGMXLIB because they are binded in GemLIB.
- DEMO program release 3.
- WindClear() : if a background color is not supported, white color is taken.
- new public variable *app.pipe* : it is a 256-buffer in global memory used by WinDom for

extern GEM communication. It can used by programer for his own communications.

- Fixed bug in FselInput (case for ext == NULL).
- Fixed bug in toolbar resizing.
- Fixed bug in window menu selection with Naes.
- Fixed bug in FormThumb().

Programming guideline of WinDom

Version 1.00 (November 2000)

- In this version, many fundamental changes appear. Please, read the section Convert your old WinDom applications if you already have used WinDom,
- WinDom is now available to Sozobon X (thanks to A.Bercegeay for his precious help),
- Source files have new structure : all new AES and new VDI functions are binded in a separated library. WinDom should be linked with GemLib pl38 if you use Sozobon version and should be linked with PCGEMLIB.LIB and PCGMXLIB.LIB (this last library is a part of WinDom package) if you use Pure C.
- Many bugs have been fixed in Event Library. It is now possible to attach severals function to a same function (see EvntAdd(), EvntRemove(), EvntEnable() and EvntDisable()). For this raison, functions EvntAttach(), EventDelete() and EvntExec() have a different behavior than in the previous version. **MU_XTIMER**, **MU_XM1** and **MU_XM2** events can be binded to a global function,
- An illimited number of data can be attached to a window (see Data Library) : the fields data and data2 in WINDOW structure are now obsolet !
- Bag bug fixed in modal formulars (this bug appeared in the last version),
- TEDINFO objects are now duplicated by ObjcDup(),
- New mode **WF_MENU_HILIGHT** in WindSet() : a function can be called when the user navigates a window menu : very usefull for help message !
- New mode **WF_ICONDRAW** in WindSet(),
- New functions ApplSet() / ApplGet() to parametrize the application,
- Bug fixed in the detection of the ApplControl() function,
- WinDom configuration file is now searched in the **$ETC** directory,
- The variable evnt.timer replaces the two variable evnt.hi_timer and evnt.lo_timer,
- New standard function, std_btm, to handle **WM_BOTTOMED** message,
- Bugs fixed in std_rtlnd() and std_dnlnd() (fixed by A.Bercegeay),
- New mode in ObjcChange() : a state can be unset (contrib of A.Bercegeay),
- Thumbs objects are now automatically handled (FormThumb()),
- New variable app.nplanes giving the number of planes of the current screen resolution (requested by A.Bercegeay),

- New messages WM_(UP|DN)(LINE|PAGE), WM_(LF|RT)(LINE|PAGE) allows to bind directly the arrowed event (WA_) without use an arrowed event function (**WM_ARROWED**),
- New functions RsrcXload(), RsrcGaddr() and RsrcGhdr(),
- MenuPopUp() has been completly rewritten by A.Bercegeay to support slider widget for large popup,
- std_dstry() sends an **AP_TERM** message if a desktop menu is not defined,
- some problems in **DCRBUTTON** object redraw have been fixed,
- when disabled, a **MENUTITLE** can contain a label (see DEMO.APP for an example).

Programming guideline of WinDom

Version of March 2000

- In this version, many fundamental changes appear. Please, read the section Convert your old WinDom applications if you already have used WinDom,
- Documentation has been translated in english. Only the english version is now supported. The documentation has been revised, many errors corrected ...
- New functions EvntAttach(), EvntDelete(), EvntFind(), EvntExec() to handle easily all GEM event. WinDom uses a new way to handle window events and all function pointers of the WINDOW structure disappear,
- Bug fixed in ConfWrite(),
- New messages **AP_BUTTON** and **AP_KEYBD**,
- **FRAME_NOBG** bit (see FrameSet()) now documented,
- with low resolution screen, the extended object text are displayed with the system font with a fixed size,
- new FormCreate() feature: if a form is bigger than the desktop, widget scrollers are automatically added to the window,
- new extended objet **DIALMOVER** (17) to draw background forms,
- new variables windom.relief.{color,mono} in configuration file,
- bug fixed in give_iconifyxywh() : it returns now a correct value with MagiC,
- the **MENUTITLE** seperator objects are better displayed,
- the message **WM_LOADCONF** is renamed to **AP_LOADCONF**,
- new FormAlert() function,
- if a menu or a toolbar is added or removed in a opened window, the size of the window is update in order to keep the same workspace size (see WindSet()),
- MouseObjc() works now correctly,
- bog fixed in MenuPopUp() : a false value was returned when the popup was closed without mouse movement,
- new function ObjcAttach() which allow to link object to variable or function,
- bog fixed in frm_fld() function (fuller event of window formular),
- new **WF_ICONTITLE** mode in WindSet() and WindGet() to define the iconified window

title, new comportement of iconify and uniconify standard event function: the window gets the icon or normal title without a specific call of WindSet(),

- new **WF_ICONDRAW** mode in WindSet() and WindGet() to define the icon redraw function,
- the *fullsize* field in the WINDOW structure is replaced by the **WS_FULLSIZE** bit in the *status* field of the WINDOW structure,
- work area of WinDom windows (toolbar and menu take into account) are now correctly clipped during a **WM_REDRAW** event (Zerkman report).

Programming guideline of WinDom

Version of June 1999

- bad bug in get_cookie() fixed (supervisor mode),
- diverses bugs fixed in WindGet(),
- the *app.ntree* variable contains the numbre of tree in the ressource file loaded by RsrcLoad(),
- a menu window can scroll (see (windom.menu.scroll),
- New function MenuScroll(),
- bug fixed in DEMO.APP,
- MenuPopUp():
  - popup placement bug fixed,
  - Zerkman suggestion: a list popup style can be parametrized in configuration file (see windom.popup.* variables),
  - new P_CHECK option in MenuPopup()
- WindClear() : new variables windom.window.bg.*,
- GrectCenter() : new prototype and use the configuration variable windom.window.center,
- new coonfiguration variables: windom.popup.window, windom.mform.attrib and windom.window.effect,
- FselInput() :
  - new prototype de FselInput() (two additional parameters),
  - nez configuration variables windom.fsel.fslx, windom.fsel.path and windom.fsel.mask,
  - BoxKite 2.00 compatibilty,
  - specific use of fslx_() functions (window file selector) if possible (MagiC, Wdialog),
  - Selectric compatibility,
- If you use new file system with lmong filename (MinixFS,Vfat,Vfat32,...) the WINDOM.CNF can be rename in .windomrc (unix style),
- Bug fixed in RsrcLoad(): ressources placed in a subdirectory are correctly loaded with Naes. The ressource pathname can have a TOS format (C:\subdir\) or a MINT format (/c/subdir/)
- Bug fixed in ApplInit(): system extensions are correctly interpreted when appl_getinfo()

function is partially implemented (WinX,Wdialog),

- New functions: conv_path(), vq_extfs(), vq_winx(), Galloc().
- If the AV environnent is installed (AvInit()), the window opened are declared to the AV-server (message AV_ACCWINDOPEN, AV_ACCWINDCLOSED). In this case, the AV server can possibly use the AV_DRAGDROP message instead of the standard AP_DRAGDROP,
- New mode in ObjcDraw()/ObjcChange(): OC_OBJC.
- BubbleGEM functions are implmented for classical GEM formulars (see functions BubbleModal(), BubbleDo() and BubbleGet()).
- The %S type in ConfInquire() function is now obsolet: ConfInquire() detect if the string have double quote deliminters. This type is keeped for compatibility,
- New fonction WriteConf().
- New fonctions vq_naes(), ApplControl(), appl_control().
- New window feature: with MagiC or Naes, a shift-closer on a window masks the application,

Programming guideline of WinDom

Version de Septembre 98

- bug en True Color avec les icônes des objets étendus enfin fixé,
- l'objet MENUTITLE fonctionne maintenant correctement sous TOS et des bugs d'affichage ont été fixé,
- bug corrigé dans la fonction frm_menu() : on pouvait cliquer des objets DISABLED,
- WindSet(WF_TOP): une fenêtre recouverte par une fenêtre modal ne peut plus être mis en premier plan, cette action s'appliquera à la fenetre modale,
- WindSet(WM_BOTTOM): une fenêtre modale active ne peut pas être mis en arrière-plan. Pour cette fenêtre, il ne se passera rien,
- FontSel(): le paramètre flags accepte le bit VSTHEIGHT et MONOSPACED,
- MagiC est testé par appl_find("?AGI") et non plus par le numéro de version de l'AES (3.99),
- nouveaux flags AES4_BEVENT et AES4_UNTOPPED de app.aes4,
- nouveau chapitre qui décrit des fonctions AES bindées (définie dans WinDom),
- nouvelles libraries de support rigoureux du protocole AV,
- nouvelles fonctions : WindAttach(), vq_magx(), has_appl_getinfo(), vq_tos(), ApplWrite(), ...
- lire la remarque sur le message AP_TERM,
- Les fonctions ObjcDraw() et ObjcChange() acceptent le flag OC_MSG (parametre depth), qui signifie que le redraw se fera par la fonction EvntWindom() et non en interne dans la fonction. (non documenté depuis la version d'Avril 98)
- Les évemenents spéciaux créé a partir d'un MU_BUTTON genère maintement un MU_BUTTON.
- vieux bogues fixés dans WindGet()/WindSet().
- sous MagiC, Fsel utilise le sélecteur de fichier de MagiC dans une fenêtre (faire attention à cela).

Programming guideline of WinDom

Version de Mai 98

- WindGet() en mode WF_WORKXYWH et WF_CURRXYWH retourne maintenant des valeurs silimaires à l'AES 4.0 lorsque la fenêtre n'est pas encore ouverte c'est-à-dire que les valeurs retournée sont celles passé à la fonction WindCreate() en mode WF_CURRXYWH ou WF_WORKXYWH modulo les attributs menu et toolbar compris (report de Pascal Ognibene).
- la fonction ConfGetLine() retourne maintenant le numéro de la ligne qui a été lue.
- bug fixé dans ConfRead(),
- bug fixé dans l'évaluation des raccourcis clavier des menus,
- le type XBOXCHAR devient SLIDEPART,
- ajout d'un nouveau type étendu MENUTITLE qui permet de mettre des entrées de menu avec des fontes quelconques et qui respecte l'alignement des raccourcis clavier, (faire attention aux nouveaux type étendus de SLIDEPART et MENUTITLE qui peut entrer en conflit avec l'ancien type XBOXCHAR),
- ajout des fonctions FontName2Id() et FontId2Name(),
- refonte totale des variables du fichier WinDom (ainsi que de APPvar), de nouveaux champs font leur apparition (voir Syntaxe du fichier windom.cnf),
- bug dans FontSel fixé (lié à utilisation des pointeurs dans la fonction),
- meilleur évaluation des raccourcis clavier dans les menus : plus de type possible et indépendance vis à vis des claviers non français, notamment on peut utiliser n'importe quelle lettre ascii,
- ajout de la fonction keybd2ascii(),
- bug fixé dans le positionnement à la souris du curseur des champs éditable; attention, cela ne fonctionne pas en justification à droite,
- Correction de la doc: plein de faute en moins, quelques chapitres ont été restructuré, lire la remarque sur les terminaisons de programme

Programming guideline of WinDom

Version de février 98

- Ajout de la fonction ConfGetLine(),
- les champs xpos, ypos, xpos_max et ypos_max de la structure WINDOW ont changé de type, ils sont maintenant de type long pour un gain de précision,
- ajout d'un second champs de donnée, data2, dans la structure WINDOW. On peut ainsi lier facilement des données à un formulaire (dont le champ data était déjà occupé par les données propres du formulaire).
- nouvelle fonction RsrcUserDraw() qui permet d'attribuer des routines de dessin quelconques aux objets, ces objets auront le type étendu 0XFF réservé donc par WinDom,
- les types étendus qui affichent du texte acceptent maintenant des attributs de texte (gras, italic, légé, creux, ombré, souligné).
- bug BubbleGEM avec les formulaires dupliqués en mémoire corrigé (report SoulFish),
- ajout de la fonction set_bit(),
- les routines standard d'icônification envoie les fenêtres icônes en arrière plan,
- mise à jour de la doc (comme d'habitude quoi).

Programming guideline of WinDom

Version de janvier 98

- réécriture de la librairie Conf pour optimiser les accès disque très lents sous (Multi)TOS, bug fixé dans la recherche du fichier WINDOM.CNF dans les différents chemins,
- bug des routines de scrolls fixé!
- ObjcString() fonctionne maintenant sur les icônes,
- routine std_allicn (AllIconify) implémenté,
- la fonction Windclose() vérifie maintenant que la fenêtre n'est pas fermé (WS_OPEN) sinon elle retourne -1,
- sous MagiC (v 5), FselInput() appelle le sélecteur de fichier de MagiC pour accèder aux noms de fichiers long. De plus le chemin "HOME=" est mis dans la liste de chemin prédéfini,
- les gadgets des frames sont maintenant correctement affichés quelquesoit la version du TOS,
- nouvelle fonction ExecGemApp(),
- ajout des fonctions FrameInit() et FrameExit().

Programming guideline of WinDom

Version de décembre 97

- La librairie Bubble prend en compte les nouveautées de la version 4 de BubbleGEM :
    - ajout du champ bubble_fntid et bubble_fntsize dans APPvar,
    - ajout du mot clef bubble_font dans WINDOM.CNF
    - Prise en compte de la variable d'environnement BUBBLEGEM (officielle) en plus de la variable BUBBLE (non-officielle, WinDom) pour localiser le programme BUBBLE.APP.
- Ajout de la fonction FontSet(), un sélecteur de fonte.
- Nouvelle version du module AES4 (binding de fonction de l'AES4) pour palier à un problème de compilation sur de vieille version de PureC.

Programming guideline of WinDom

Version de novembre 97

- Ajout du message WM_LOADCONF
- Ajout de la librairie Configuration.
- Ajout de la variable globale windom_version.
- Ajout du champ windom_version dans le fichier de configuration.
- Frame: les frames ont maintenant leur propres attributs (reste encore quelques bugs).
- B_UNTOPPABLE: correction d'un bug (reste a vérifier sur vieux TOS)
- ajout du champs untopped dans la structure WINDOW. Cette fonction gère le message WM_UNTOPPED. Sur les systèmes ne gérant pas ce message, il est émulé (dans une certaine mesure: WinDom ne peut prendre en compte que ses propres fenêtres).
- Les entrées des menus peuvent être de type quelconque.
- correction d'un bug d'EvntWindom() pour la gestion des formulaires apparut lors d'une modif antérieure.

Programming guideline of WinDom

Version de août 97

- Formulaire multiple: ajout du bouton onglet (MLTIFRM).
- Ajout du bouton KPOPUPSTR pour donner des raccourcis claviers aux popup.
- Ajout d'un fichier de configuration des applications WinDom
- Ajout de nouveaux champs dans APPvar pour gérer la couleurs des objets étendus.
- disparition des champs test, keybd et button dans APPvar remplacé par le champ flag organisé en champs de bits. Les bits libres permettront une extension du système.
- ajout des champs m1 et m2 dans WINDOW correspondant aux message MU_M1 et MU_M2.
- MenuPopUp() est en pleine mutation mais le codage n'est pas terminé
- bugs fixés dans les routines de gestions des raccourci claviers et des champs éditables (elles ne prenaient pas en compte les objets HIDETREE)

Programming guideline of WinDom

Version de mai 97

- ajout de la librairie frame,
- ajout de la librairie bubble,
- ajout du champ extramsg dans la structure WINDOW,
- debugguage divers.

Programming guideline of WinDom

Compiling a WinDom Application

First utilisation

Before compiling your first WinDom application, you should install the library in your environmment.

Copy from the WinDom package:

```
windom\include\windom.h
```

and optionally :

```
windom\include\scancode.h
windom\include\av.h
```

in the **include** folder of your compiler. Then copy the library itself from the WinDom package :

- windom\lib\purec\pcgemlib.lib if you use Pure C
- windom\lib\gcc\libwindom.a if you use Gcc 2.9.xx or newer
- windom\lib\gcc281\windom.olb if you use Gcc 2.8.1
- windom\lib\sozobon\windom.a if you use Sozobon X

in you **lib** folder of you compiler.

Notice Gcc 2.9 runs only with MiNT with an Unix file system hierarchy. For people using plain-TOS or MagiC, gcc 2.8.1 is a good choice and works fine.

Pure C users have to install PCGMXLIB library. Copy from WinDom package :

```
windom\pcgemx\include\pcgemx.h
windom\pcgemx\lib\pcgmxlib.lib
```

respectyvely in the **include** and **lib** folders of Pure C folder.

Sozobon users and Gcc users have to use MGemlib pl39 or GemLib pl40 or newer versions. These libraries are respectively available on (!url [http://gemtos.free.fr] [http://gemtos.free.fr]) and (!url [http://www.freemint.de] [http://www.freemint.de]).

Compiling:

You just have to include the WinDom header file in your source files :

```
#include <windom.h>
```

For compiling and linking with Sozobon and Gcc use the following makefiles :

```
# Makefile for compiling with MgemLib pl39
CC = gcc  # for Gcc user
CC = cc   # for Sozobon user
CFLAGS = -O -DUSE_MGEMLIB
LDFLAGS = -lwindom -lmgem
# eof

# Makefile for compiling with GemLib pl40
CC = gcc  # for Gcc user
CC = cc   # for Sozobon user
```

```
CFLAGS = -O -DUSE_GEMLIB
LDFLAGS = -lwindom -lgem
```

If any of switches USE_MGEMLIB or USE_GEMLIB are specified, USE_MGEMLIB is used by default.

With Pure C, you have to use the library PCGMXLIB.LIB provided in the WinDom package. This library complets the GEM library. You also have to link with PCSTDLIB and PCTOSLIB. (M)Gemlib has not been tested with Pure C. However, if you want use it, you have to recompile WINDOM.LIB.

```
;; Pure C projet file (standard)
windom.lib
pcgemlib.lib
pcgmxlib.lib
pcstdlib.lib
pctoslib.lib
```

Recompiling the library:

In the following part, '% ' designs the prompt of your shell interpreter.

- unzip the windom package in a folder and keep only the folders **src** and **include** :

```
% unzip wndm0109.zip -d windom
% cd windom/src
```

- evntually, edit options.h and change some compilation options.
- With Pure C, use the project file windom.prj to compile WinDom.
- With other compilers (gcc, sozobon), you have to use a shell interpreter (tcsh, mupfel, ...) and use a make or gmake program. You have to choice between GEMLIB (pl40) and MGEMLIB (pl39).
- With Gcc 2.8.1, adapte the makefile m_gcc281 to your environnement or sets the following environnement variables :
  - CC_GCC281 to 'gcc' or complete path of gcc if it is not defined in your PATH variable,
  - AR_GCC281 to 'ar' or complete path of ar if it is not defined in your PATH variable,
  - GNUINC to the path of gcc include folder,
  - GNULIB to the path of gcc lib folder.
- With Gcc 2.9.5, compiler uses standard unix hierarchy ie /usr/include, /usr/lib and /usr/GEM/include. If your environnement, is different, adapte the makefile m_gcc to your environnement.
- With Sozobon X, adapte the makefile m_sox to your environnement or sets the following environnement variables :
  - CC_SOX to 'cc' or complete path of cc if it is not defined in your PATH variable,
  - AR_SOX to 'ar' or complete path of ar if it is not defined in your PATH variable,
  - SOXINC to the path of sozobon include folder,
  - SOXLIB to the path of sozobon lib folder.
  - SOXBIN to the path of sozobon bin folder.
- type make in the shell. It returns :

```
Targets are gcc, gcc281, soz :
make gcc : compile for gcc 2.9.5 (for MiNT with an Unix file system hierarchy)
make gcc281 : compile for gcc 2.8.1 ( for other systems)
make soz : compile for Sozobon X
```

So select your target and start the compilation.

Programming guideline of WinDom

A tutorial of Windom step by step ...

Some conventions are used in this documentation:

- Bold face is used for constants, e.g. **VSLIDER**,
- Italic face for variables or parameters, e.g. *app*.*x*,
- Function names have trailing parenthesis. E.g. WindCreate().

Programming guideline of WinDom

Create a window

Let's start this tutorial with a very basic example: create and handle a window in the WinDom environment. Look at this first example:

```
#include <windom.h>

void main( void) {
     /* window descriptor */
     WINDOW *win;

     /* WinDom initialisation */
     ApplInit();

     /* Create a window and keep its descriptor */
     win = WindCreate( NAME|MOVER|CLOSER,
                       app.x, app.y, app.w, app.h);

     /* Open the window */
     WindOpen( win, app.x, app.y, app.x+app_w/2, app.y+app.h/2);

     /* Handle the GEM events */
     while( wglb.first)
          EvntWindom( MU_MESAGE);

     /* Terminate Windom session */
     ApplExit();
}
```

As we said in the introduction, WinDom programming is very similar to GEM programming. Actually, it is possible to use WinDom exactly like GEM but it is not the most efficient way. The only difference between GEM and WinDom in this example is the GEM events returned by EvntWindom() (the equivalent of evnt_multi()) are not handled! In fact, events are implicitly handled by EvntWindom() by using standards functions given by WindCreate() to the window *win*. The variable *win* is the window descriptor.

For example, WindCreate() gives to the new window the function WindClear() (this function draws a white bar inside the window) as redraw event function. It means that when EvntWindom() catches a redraw event (ie the GEM message **WM_REDRAW**), it will use the function WindClear() to refresh the window's workspace. Obviously, the first step in WinDom programming is to write a new redraw function for the new window created by WindCreate().

What's about the event handling? In this example, the while() instruction uses the WinDom global variable *wglb* which gives information about windows in WinDom. The field *wglb.first* points to the descriptor of the first window created by WindCreate(). If this variable is **NULL**, then there is no more window. And why there is no window? Because the window previously opened has been closed by the user (by cliking the closer widget of the window). When you click on the closer widget, the AES screen manager sends a **WM_CLOSED** message to our application. Of course, our window own a standard function handling this message. This function sends a special message to the screen manager (the **WM_DESTROY** message). This message is not a standard GEM message but a special WinDom message. In WinDom, we make the distinction beetween close a window and destroy a window. **WM_DESTROY** means that the window and its data have to be

destroyed. The standard function attributed by WindCreate() to the message **WM_DESTROY** closes and deletes the window using the WindClose() and WindDelete() WinDom functions. The destroy function is usually the second function (after the redraw function) written by the developper.

Programming guideline of WinDom

The redraw function of a window

Remember that the redraw function of a window is just a function attached to the **WM_REDRAW** GEM message. Let's have a look at how to create and declare such a redraw function.

Programming guideline of WinDom

General rules

A redraw function (as every event function) has the following structure:

```
void a_redraw_function ( WINDOW *win) {
        ...;
}
```

The parameter *win* is the descriptor of the window that has to be redrawed. Let's give some important remarks to design this redraw function.

- Fist of all, one should not clip anything (i.e. use the VDI clipping functions) inside the redraw function. The clippping zone is handled by EvntWindom() (by using the AES rectangle list).
- When you perform VDI calls, you can use the VDI virtual workstation opened by WindCreate(). The handle of this workstation is given by the variable *win->graf.handle*.
- The content of a window often depends on the sliders positions and the size of the window. For that purpose, several field of the window descriptor are devoted to help you managing sliders. These are *win->xpos, win->ypos, win->xpos_max* and *win->ypos_max*. For more details, read the section about the (!url [sliders] [Window sliders]).
- By convention, the first thing to do in the redraw function is to call the function WindClear() to draw the background. This function can be parametrized by the user.

In the next section, we'll see some examples that create windows. You'll see also how to attach data to a window using the Windom DataAttach() funciton.

Programming guideline of WinDom

proportional window

In this example, we want to open a window which containts a circle. In this case, sliders are not used. The data concerning this window is stored in a user structure and attached to the window. This provides us with some usefull informations about the window content (color, pattern, ...).

```
struct circle {
     int color;
     int pattern;
};
```

And now, the function in charge of destroying the window:

```
void Destroy( WINDOW *win) {
        struct circle *C;

        C = (struct circle *) DataSearch( win, 'CIRC');
        free( C);                   /* free memory */
        DataDelete( win, 'CIRC');
        WindClose( win);            /* close the window */
        WindDelete( win):           /* delete the window */
}
```

The redraw function:

```
void Draw( WINDOW *win) {
int x, y, w, h;
        struct circle *circ;

        circ = (struct circle *) DataSearch( win, 'CIRC');
        /* Get the workspace coordinates */
        WindGet( win, WF_WORKXYWH, &x, &y, &w, &h);
        /* Clear the background */
        WindClear( win);
        vsf_color( win->graf.handle, circ->couleur);
        vsf_interior( win->graf.handle, circ->motif);
        v_circle( win->graf.handle, x+w/2-1, y+h/2-1, min( w, h)/3);
}
```

The main function is:

```
#define WIN_CIRCLE      1

        int main( void) {
                WINDOW *win;
                struct circle *C;

                /* Init WinDom */
                ApplInit();

                /* Create a circle data */
                C = (struct circle *) malloc( sizeof( struct circle));

                /* Create the window */
                win = WindCreate( WAT_ALL, app.x, app.y, app.w, app.h);

                /* Attach data to window */
                DataAttach( win, 'CIRC', C);
```

```
        win->type = WIN_CIRCLE;

        /* Declare new event message */
        EvntAttach( win, WM_REDRAW, Draw);
        EvntAttach( win, WM_DESTROY, Destroy);

        /* Open the window */
        WindOpen( win, app.x, app.y, app.w/2, app.h/2);
        WindSet( win, WF_NAME, "titre");
        WindSet( win, WF_INFO, "infos");

        /* Handle GEM event */
        while( wgbl.first)
                evnt_windows( MU_MESAG);

        ApplExit();
        return 0;
}
```

To create a new kind of window we needed to:

- write the redraw function,
- write the destroy function,
- attach data to window.

This is the general way to create new window in a WinDom application. Note that the field *type* of the window descriptor is not used. It is just used to identify the nature of data attached to the window. In our case, the window is of type **WIN_CIRCLE**. Data are attached to the window using the function DataAttach(). Then data are recovered (for example, in the redraw function) using the function DataSearch().

To declare a new event function, we use the function EvntAttach(). It is a very important function in WinDom. Thus the call:

```
EvntAttach( win, WM_REDRAW, Draw);
```

give to the window *win* and the message **WM_REDRAW** the function Draw().

Try to compile this example and see how it works. If you close the window, the application exits. This example doesn't use any slider but in the next one, we are going to see how to handle them.

Programming guideline of WinDom

Non proportional window

Let's start using sliders. There are two steps:

- initialize sliders related variables in the window descriptor,
- write a redraw function (of course).

Get an usefull example: display a text in a window.

First, we have to create a data structure that we will attach to the window using DataAttach(). Here it is:

```
typedef struct {
        char *buffer; /* address of text */
        char **line;  /* table of each lines */
        int maxline;  /* number of line */
        int wchar,hchar; /* size of a character */
} TEXT;
```

We suppose that we are able to load a text in memory and that each line of the text is terminated by a null-byte.

The variables *xpos* and *ypos* of the window descriptor represent the position of data inside the window. In this case, *xpos* is the first displayed column and *ypos* the first displayed line. The variable *w_u* and *h_u* give the width and the height in pixel to shift when scrolling the window. These values are the width and the height of a character (the font used to display the text is supposed non-proportional). These values are also used to compute the size of the sliders.

Now, write the function creating a text window:

```
WINDOW *OpenText( TEXT *text) {
        WINDOW *win;
        int attrib[10];

        win = WindCreate( WAT_ALL, app.x, app.y, app.w, app.h);
        EvntAttach( win, WM_REDRAW, draw_text);
        DataAttach( win, 'TEXT', text);
        /* Maximal lenght of a line */
        win -> xpos_max = 255;
        /* Number of line */
        win -> ypos_max = text->maxline;
        vqt_attributes( app.handle, attrib);
        /* Height of a cell character */
        win -> w_u = attrib[8];
        /* Width of a cell character */
        win -> h_u = attrib[9];
        /* Height of a character */
        text-> wchar = attrib[6];
        /* Width of a character */
        text-> hchar = attrib[7];
        /* Open the window */
        WindOpen( win, app.x, app.y, app.w, app.h);
        /* Update the size and position of sliders */
        WindSlider( win, HSLIDER|VSLIDER);
        return win;
}
```

We write now the redraw function. The algorithm is:

- draw the background,
- draw line visible in the workspace window ie from *ypos* to the last line visible in the clipping zone,

- draw each line from the first visible column (i.e. *xpos*).

```
void draw_text( WINDOW *win) {
        int x,y,w,h;
        int hcell, hcar;
        int i, attr[10];
        TEXT *ptext = (TEXT *)DataSearch( win, 'TEXT');

        /* Get some usefull information */
        WindGet( win, WF_WORKXYWH, &x, &y, &w, &h);
        w += x-1;
        vqt_attributes( win->graf.handle, attr);
        hcell = attr[9];
        hcar  = attr[7];

        /* Background */
        WindClear( win);

        /* Foreground */
        vswr_mode( win->graf.handle, MD_TRANS);
        /* vertical offset for a nice text drawing */
        h = hcell - hcar;

        /* from the first line visible to the end ...
         * Convention: we have always 0 <= win->ypos < win->yposmax */
        for( i=win->ypos; i<win->ypos_max ; i++)  {
                y += hcell;

                /* If the line is upper the clipped zone? */
                if( y < clip.g_y)
                continue;

                /* line inside the window ? */
                if( strlen( ptext->line[ i]) > win->xpos)
                        v_gtext(win->graf.handle, x, y - h, ptext->line[i] + win->xpos);

                /* End if the line is downer the clipped zone */
                if (y > min( w, clip.g_y + clip.g_h-1))
                        break;
     }
}
```

As EvntWindom() clips all screen output using the AES rectangle list, we just have to use the WinDom global variable *clip*. This variable (a GRECT structure) containts the coordinate of the zone clipped by EvntWindom()) during a redraw event. This variable allows you to decrease the complexity of the redraw function.

This case illustrates a common case of sliders use. Some cases are more complex (for example a window displaying icons like the ideal window of the GEM desktop). The (!url [sliders] [Window sliders]) section is a detailed description of sliders use.

Programming guideline of WinDom

Destroy a window

When you create a new kind of window, such as a text window, you need to load, reserve memory and attach the data to the window, then you write a specific redraw function. When the work is done, you'll need to close, delete windows and cleanup memory. For that purpose WinDom supports two messages:

- WM_CLOSED - the window should be closed at screen,
- WM_DESTROY - the window should be delete and data freed.

Of course every newly created window gets by default standard functions that handle these messages.

When should I destroy a window ?

Usually, the destruction of a window is due to a GEM event message, for example the user clicks on the closer widget, then the screen manager of the AES reacts by sending a message to the application a **WM_CLOSED** message. Of course WindCreate() attributes to the WinDom a standard (i.e. a predefined) function handling this message. So you don't need to close and destroy explicitly a window (with the functions WindClose() and WindDelete()), because the standard functions perform this task for you.

The only case when you should destroy explicitly a window is when your window has no closer function and no destroy function or when your application exits.

The standard closer function:

This function (all standard functions of WinDom are located in the file SRC/STDCODE.C of the WinDom package, the standard closer function is std_cls()) just sends an another message to the application window, a **WM_DESTROY** message. This predefined WinDom message, means that the window should be deleted.

When should I write a new closer function ?

As soon as you want a window to keep its attributes until it is opened again. Such a window has to just disapear from the screen stay in memory when it is closed. Many GEM programs provide this feature: a window is closed but its data is kept. Closed windows are accessible from the application menu or by an icon on the application desktop. Then the new closer function is something like that:

```
void new_std_cls( WINDOW *win) {
        /* Keep information somewhere ... */
        insert_in_menu( win->data);
        /* ... and close the window */
        WindClose( win);
}
```

Redefining this closer function, the window will never be deleted. That's why you'll need to destroy it explicitly before the end of the application. This results a code similar to the following:

```
int main( void) {
        WINDOW *win;

        ApplInit();
        /* Create windows ... */
        win = WindCreate(...);
        /* Attribute the new function */
        EvntAttach( win, WM_CLOSED, new_std_cls);

        ...

        /* Delete all windows */
        while( wglb.first) {
#if GOODWAY
                ApplWrite( wglb.first, WM_DESTROY);
                EvntWindom( MU_MESAG);
#else /* Very bad way */
                WindDelete( wglb.first);
#endif
        }

        /* then quit ... */
        ApplExit();
        return(0);
}
```

As you can see, we delete all windows using the WinDom global variable *wglb.first*. The best way is to send a destroy message then execute all GEM event with EvntWindom() rather than deleting the window with WindDelete(). Indeed, some AES present the following problem: if you create/delete too many windows in same time, the system may crash.

Remarks: In a old version of WinDom, the documentation shows this example using the message **WM_CLOSED** instead the message **WM_DESTROY**. This method had two inconvenient:

- the window needs to use the standard closer function or a similar user function,
- for old TOS version, WinDom emulates the iconification of WinDom, the smaller wigdet of window is simulated by clicking the close widget while the control key is pressed. Now, imagine an application with a menu containing a shortcut with a control key, for example:

  ```
  Close Window    ^U
  ```

  When you type the key shortcut, the window is closed but as the control key is pressed, WinDom interprets it as an iconification request.

The standard destroy function:

This function is called by EvntWindom() when it gets an **WM_DESTROY** message. This function destroys explicitly the window by calling both WindClose() and WindDelete(). Of course, in the previous example, the destroy window should be rewritten by something like that:

```
void new_std_dty( WINDOW *win) {
        /* Get my data */
        void *mydata = DataSearch( win, 'DATA');
        /* Save data if needed */
        if( needed) save_my_data( win->data);
        /* free memory ... */
        free_my_data( mydata);
        DataDelete( win, 'DATA');
        /* ... and delete the window */
        WindDelete( win);
}
```

This example is very special because we have changed the standard closer function. Usually, we don't change this function, and the destroy function should be:

```
void new_std_dty( WINDOW *win) {
        /* Get my data */
        void *mydata = DataSearch( win, 'DATA');
        /* Save data if needed */
        if( needed) save_my_data( win->data);
        /* free memory ... */
        free_my_data( mydata);
        DataDelete( win, 'DATA');
        /* ... close the window ... */
        WindClose( win);
        /* ... and delete the window */
        WindDelete( win);
}
```

If needed, it is a good idea to save data inside the destroy function before freeing it.

Tip: WindDelete() closes the window with WindClose() if it is not already closed. Thus, calling WindClose() before WindDelete() is optional.

Programming guideline of WinDom

Terminate a WinDom application

There is no unique way to terminate an application, but we give here some examples to quit in a clean manner. Proceed in three steps: quit the main event loop, close and delete all windows and clean up memory and other ressources.

The end of a program may occur in various situations:

- the user selects the Quit options of the menu application: you have to inspect the **MN_SELECTED** or **WM_MNSELECTED** messages.
- all windows are closed and your application has no global menu. This should terminate the application and can be detected with the WinDom variable *wglb.first* being **NULL** (this means that all windows are closed and deleted).
- the application received an **AP_TERM** message: this message means that the application should terminate now.
- the application produced an error and received a MiNT crash signal.

The better method is to write a function which terminates properly the application. It typically looks like that:

```
void ap_term_( void) {
        /* Close all windows: see the previous section */
        while( wglb.first) {
                ApplWrite( wglb.first, WM_DESTROY);
                EvntWindom( MU_MESAG);
        }
        /* Free all ressource */
    /* if you have install extended object type ...*/
        RsrcXtype( 0, NULL, 0);
        /* ... and free the ressource */
        RsrcFree();

        /* Others ressources to free */
        ...
        /* Quit WinDom environment */
        ApplExit();

        /* Finish Application */
        exit( 0);
}
```

Now your application should handle the message **AP_TERM**. As we nowadays have multitasking OS, handling this message is a general rule for any GEM application. You can attribute the ap_term() function to this message like that:

```
        EvntAttach( NULL, AP_TERM, ap_term);
```

Then the ap_term() function is invoqued when EvntWindom() recieves an AP_TERM message.

You can deal with your application crashing by trapping the MiNT signals sent with the function

Psignal():

```
        Psignal( SIGQUIT, ap_term);
        Psignal( SIGBUS,  ap_term);
        etc ...
```

Note: MiNT signals are available with MagiC.

If your application does not install a desktop menu, the main function may look like:

```
int main( void) {

        ApplInit();
        EvntAttach( NULL, AP_TERM, ap_term);
        Psignal( SIGQUIT, ap_term);
        /* ... others signals ... */

        /* Main loop event */
        while( wglb.first) EvntWindom( MU_MESAG);
}
```

And if your application has a desktop menu:

```
/*
 * This function manages the desktop menu
 * The evnt.buff variable is a WinDom global variable
 * that contains the AES buffer message returned by
 * evnt_multi() after a MU_MESAG event.
 */

void do_menu( void) {
        int title = evnt.buff[3];

        switch( evnt.buff[4]) {
        case QUIT:
                ApplWrite( NULL, AP_TERM);
                break;
        }
        MenuTnormal( NULL, title, 1);
}

int main( void) {
        OBJECT *menu;

        ApplInit();
        /* Install the menu */
        RsrcLoad( "myrsc.rsc");
        rsrc_gaddr( 0, DESKTOP_MENU, &tree);
        MenuBar( tree, 1);
        EvntAttach( NULL, AP_TERM, ap_term);
        Psignal( SIGQUIT, ap_term);
        /* ... others signals ... */
        /* trap the menu selections */
        EvntAttach( NULL, MN_SELECTED, do_menu);

        /* Main loop event */
        while(1) EvntWindom( MU_MESAG);
}
```

Remark 1:

The EvntAttach() is more powerfull than, for example, a simple test on the message gets by EvntWindom() :

```
                do
```

```
        EvntWindom( MU_MESAG);
while( evnt.buff[3] != AP_TERM)
apterm();
```

In this case, the action of ap_term() is local. With the EvntAttach() method, the ap_term() function will be always called by any EvntWindom() invocation. It is more global because some WinDom functions call EvntWindom() and the ap_term() function may have then to be invoked. It is the case with the font selector or the popup menu manager.

More about AP_TERM

This message indicates:

- a system shutdown, in this case, the sixth word of the AES message buffer containts the value of **AP_TERM**(50),
- a screen resolution change, in this case, the sixth word of the AES message buffer containts the value **AP_RESCHG**(57),
- a user terminaison request (eg via a taskapp bar), in this case the sixth word of AES message buffer is different of **AP_TERM**(50) and **AP_RESCHG**(57).

After a system shutdown or a screen resolution change, if your application cannot finish, you have to inform AES by sending a **AP_TFAIL**(51) message by using the shel_write() function with the mode=**SWM_NEWMSG**(9). At beginning of your application you have to inform AES that your application understands the AP_TERM message like that:

```
if( has_appl_getinfo()) {
        int val1, dum;

        appl_getinfo( 12, &val1, &dum, &dum, &dum);
        if( val1 & 0x8) shel_write( 9, 1, 1, NULL, NULL);
}
```

Programming guideline of WinDom

More about events

WinDom uses an original method to handle the GEM events. It is the EvntWindom() function which performs that. First, EvntWindom() distinguishes the events applied to windows and the events applied to the application. For example, WM_REDRAW is a window event and AP_TERM is an application event.

When EvntWindom() receives an GEM event. It looks for this event into an event list. There is one event list for the application and one event list for each window. When EvntWindom() finds the event among these event lists, it executes the function corresponding to the event in the list. So, to attribute a function to an event (ie to add an element in a event list), we use the function EvntAttach().

The application event list is empty by default but it is a good thing to handle the AP_TERM and the VA_START messages. A window is created with a default event list.

The function EvntAttach() allows you to add a handler in a list. The handler may deal with event messages or other GEM events such as MU_BUTTON, MU_TIMER, MU_M1 and MU_M2. See the EvntAttach() function for more details. Note that it is possible to target a specific window or the all application with the same event. For example the following call

```
EvntAttach( NULL, WM_XBUTTON, AppButton);
```

tells EvntWindom() to call the AppButton() function when a MU_BUTTON event occurs. And the call

```
EvntAttach( win, WM_XBUTTON, WinButton);
```

tells EvntWindom() to call the WinButton() function of the window *win* when a MU_BUTTON event occurs and the window *win* is active. The two calls:

```
EvntAttach( NULL, WM_XBUTTON, AppButton);
EvntAttach( win, WM_XBUTTON, WinButton);
```

are possible.

Others functions from the Event library allows you to control events lists. The function EvntDelete() removes an event form a given list, the function EvntFind() finds an event and the function EvntExec() executes the function attached to a given event.

Programming guideline of WinDom

The window color palette

Programming guideline of WinDom

How WinDom uses the color palettes

WinDom supports an automatic color palette handling. Each window has its own palette. When a window is topped, EvntWindom() applies to the screen the window specific palette located in the field *win->graf.palette*. It is a table containing groups of 3 words the number of which depends on the screen résolution. It can be 2, 4, 16 or 256. For other resolutions (i.e. when the global variable *app.nplane* contains a value higher than 8), VDI doesnot use a color palette and the WinDom disable its color palette handling. It is the case with 16-plane, 24-plane and 32-plane resolution.

A window palette can be **NULL**. In this case, EvntWindom() uses the application desktop palette, located in the field *app.palette*. If any window belonging to another application is topped, EvntWindom() applies the desktop palette. The ApplInit() initialises the desktop palette using the current palette when the application is launched.

Programming guideline of WinDom

Create a new palette

If you want create a specific palette, you need to reserve memory for it and initialize it. That's all. There are two usefull functions devoted to palette manipulation:

```
    void w_getpal( W_COLOR *palette)
    void w_setpal( W_COLOR *palette)
```

The first function copies the current palette (the palette currently used by the display) inside the *palette* variable. The second function applies the palette *palette* at the screen.

Following is an example to create and attribute a palette to a window:

```
void create_palette( WINDOW *win) {
        W_COLOR *palette;

        /* 1) reserve memory */
        palette = (W_COLOR*)malloc(app.color*sizeof(W_COLOR));
        /* 2) initialize the palette with the current one */
        w_getpal( palette);
        /* 3) link the palette to the window */
        win->graf.palette = palette;
}
```

Don't forget to free the memory when the window is destroyed:

```
/* Destroy function */
void destroy( WINDOW *win) {
        WindClose( win);
        free( win->graf.palette);
        WindDelete( win);
}
```

Of course, WinDom uses the standard VDI palette format: the three words stand for primar components (red, green, blue), and each component has a value between 0 and 1000.

Programming guideline of WinDom

Disabling the palette handling

The palette handling, performed by EvntWindom(), can be disabled by the following call:

```
ApplSet( APS_FLAG, FLG_NOPAL, TRUE);
```

This option can be usefull when you write a desktop accessory. Suppose that a WinDom desktop accessory is launched, it use the current palette to initialize its application palette, but if the default palette was changed by a second accessory, for example, Xcontrol, when our WinDom desktop accessory will open a window, the weired palette will be restored. That's why your application should disable the palette handling in such cases.

If your accessory really needs its own palette, you should get the real value of the default palette.

Note that it is easy to modify the application palette with the current palette like that:

```
vs_getpal( app.palette);
```

Programming guideline of WinDom

The window sliders

Programming guideline of WinDom

How WinDom uses the window sliders

To correctly use sliders, you just need to initialize some variables of the window descriptor. If the size of the displayed data changes over time, you need to update the values of these variables: *xpos*, *xpos_max*, *ypos*, *ypos_max*, *h_u* and *w_u*. Some of them can be used by the window redraw function.

xpos
: indicates the horizontal position of data inside the window workspace. This variable is used to compute the position of the horizontal slider,

xpos_max
: gives the highest value of *xpos*. Actually we always have: 0 <= *xpos* < *xpos_max*,

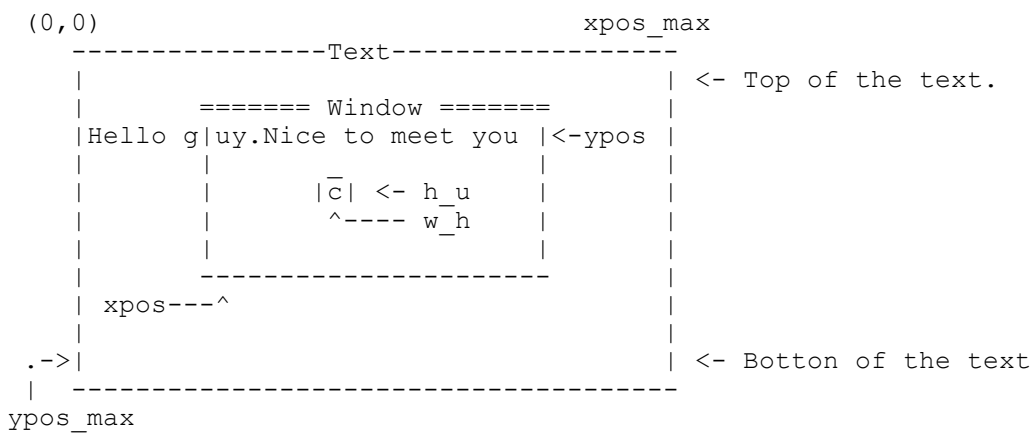
ypos and ypos_max
: are indentical to *xpos* and *xpos_max* but address the vertical slider.

h_u
: is the vertical scrolling unit (in pixel) when a vertical line scroll occurs,

w_u
: is the horizontal scrolling unit (in pixel) when a horizontal line scroll occurs,

AES uses a values between 0 and 1000 for the slider position and size. WinDom uses a values in the interval [0, xpox_max[or [0, ypos_max[. Sliders size is automatically computed depending the quantity of data displayed in the window.

Take an example. We want display an ASCII text in a window. In this context, the variable *ypos* is exactly the index of the first line displayed in the window and the variable *xpos* is the first column displayed in the window (we suppose we use a non proportional font to display the text). So the variable *ypos_max* represents the number of lines of the text and the variable *xpos_max* should the size of the largest line or a fixed number like 255, to have it easyer. The variable *h_u* represents the height of a character cell and the variable *w_u* is the width of a character cell. When the window is opened, the variables *xpos* and *ypos* should be zero. The following figure picture the situation.



This variables are set when the window is created. Then, the function WindSlider() sets the size and position of the sliders according to the variables previously described. Usually, the function WindSlider() is used when you change the value of a slider variable. In other cases, the event standard function calls WindSlider(). The previous subsection Non proportional window gives a complet example with sliders.

Programming guideline of WinDom

"Ideal" windows

The "ideale" windows changes the sliders size and position depending on their workspace size as the GEM desktop does. With the GEM desktop example, the number of icons per line displayed in a window depends on the window width, i.e. the variable *xpos_max* depends on the window width. So when the window gets a new size (with **WM_SIZED** and **WM_FULLED** messages) the variable *xpos_max* must be refreshed and the function WindSlider() called. That means that the functions attached to **WM_SIZED** and **WM_FULLED** should be customized:

```
void ideal_szd( WINDOW *win) {
    void std_szd( WINDOW *);

    win->xpos_max = <new value>;
    win->xpos = <new value>;
    /* a new value for xpos_max implies a new value for ypos_max */
    win->ypos_max = <new value>;
    win->ypos = <new value>;
    std_szd( win);      /* do not invent the wheel ! */
}
```

without forgeting to attach this function to the window with something like:

```
EvntAttach( win, WM_SIZED, ideal_szd);
```

As you can see, the Windom intern managing of sliders is robust: you just have to write two functions and use the standard functions. Have a look at these standard functions std_szd() and std_fld() in the source code (file STDCODE.C). These functions are quite complex because they support three kind of situation.

Programming guideline of WinDom

Window iconification

Window iconification is automatically supported by WinDom (via EvntWindom() function). The only thing to do (for the programmer) is to define the SMALLER widget when the window is created (call of WindCreate()) and that's all. One of cool WinDom features is the iconification works for any TOS version. To control some visal aspects of iconification (icon title and icon disply) you have to read the two last sections (Drawing the icon windows, Icon title).

However, if you want control window iconify in a different way then WinDom, you have to read the following sections.

Programming guideline of WinDom

How Windom handles iconification?

WinDom handles window iconification in the same way then AES by using three messages: WF_ICONIFY, WF_UNICONIFY, WM_ALLICONIFY and the function WindGet() and WindSet().

The iconification works with any AES version (even with your old Atari-ST computer). When AES supports iconification, Windom uses directly the AES functionality, when AES does not support iconification, WinDom emulates it. In this case, the iconification is local. In order to have a global iconfication system, WinDom uses the ICFS protocol (Iconify Server by Dirk Haun) to place on the desktop the icon window.

When the iconifier window widget is not available, WinDom uses a special combinaison to emulate it. A click on the closer window widget with the SHIFT key (right or left) pressed sends a WM_ICONIFY message (an iconify request) to the window. A click on the closer window widget with the SHIFT and CONTROL keys simulatly pressed sends a WM_ALLICONIFY message (an iconify all windows request) to the window. This emulation is done by the EvntWindom() function when it receives the AES message WM_CLOSED.

Programming guideline of WinDom

The iconification messages

When EvntWindom() receives an iconification message, it calls the associated function. The function WindCreate() attributes a set of standard functions (that allows window to handle automatically the iconification). Let's explain the signification of each message and what do the standard functions attributed to these messages.

Structure of the message:

evnt.buff[0] = WM_ICONIFY, WM_UNICONIFY, WM_ALLICONIFY
evnt.buff[1] = AES application identifier of sender
evnt.buff[2] = always 0
evnt.buff[3] = window handle
evnt.buff[4] = x coordinate
evnt.buff[5] = y coordinate
evnt.buff[6] = width
evnt.buff[7] = height

- WM_ICONIFY:
  - signification: a window should be iconified at position (x,y,w,h).
  - probable origin: Depends on system:
    - iconifier wigdet was clicked,
    - closer widget was clicked with the SHIFT key pressed,
    - manual emission of the message.
  - standard function: std_icn() iconifies the window using WindSet().
- WM_UNICONIFY:
  - signification: an icon window should be uniconified at position (x,y,w,h).
  - probable origin: the user has double-clicked on the window workspace (or manual emission of the message).
  - standard function: std_unicn() uniconifies the window using WindSet().
- WM_ALLICONIFY:
  - signification: all windows should be iconified in a unique icon at postion (x,y,w,h).
  - probable origin: Depends on system:
    - iconifier wigdet was clicked with the CONTROL key pressed,

        - closer widget was clicked with the SHIFT and CONTROLS keys depressed,
        - manual emission of the message.
    - standard function: std_allicn() closes all windows except one. This one is iconified at position (x,y,w,h).

Programming guideline of WinDom

The WindGet()/WindSet() functions

As we said, WinDom uses iconification in the same way than AES. So the standard iconifier functions use the WindSet(), WindGet() functions. If you want to customize the way your windows are iconified, you have to use these two functions. The other cases or usually fully handled by the standard functions. Let's describe these functions.

```
WindSet( win, WF_ICONIFY, x, y, w, h);
```

This call iconifies the window *win* at the position (x,y,w,h) on the screen. These coordinates are provided by the message WM_ICONIFY (or WM_UNICONIFY). If you want to iconify a window in response to another event, then AES doesn't provide you with an icon position. You can use the function

```
void give_iconifyxywh( int *x, int *y, int *w, int *h);
```

which provides you with an admissible position to iconify a window. When running under AES 4.1 (MultiTOS 1.1), there is no way to guess this position, so the function returns a constant position.

```
WindSet( win, WF_UNICONIFY, x, y, w, h);
```

This function uniconifies an icon window standing at position (x,y,w,h). These four values are provided by the message WM_UNICONIFY or by the function WindGet(win, WF_UNICONIFY, &x, &y, &w, &h).

```
WindSet( win, WF_UNICONIFYXYWH, x, y, w, h);
```

Sets the uniconified coordinates of an inconifed window. This call is useful when you create a window that is already iconified. Then you have to tell the system the uniconified coordinates of this window.

AES 4.1 does not implement the following calls. So WinDom emulates its (for a forward compatibility), the remark is true with the **WF_BOTTOM** message too).

```
WindGet( win, WF_ICONIFY, &icon, &w, &h);
```

Gives informations about a window. The variable *icon* is zero if the window is not iconified and a diferent value else. *w* and *h* give the width and height of the icon. the standard value is 72x72. When WinDom controls the iconification, this size can be customized with *app*.*wicon* and *app*.*hicon* global variables.

```
WindGet( win, WF_UNICONIFY, &x, &y, &w, &h):
```

gives the uniconified position of an icon window.

WinDom makes the difference between the name of the normal window and the name of the icon window. The **WF_ICONTITLE** mode of WindSet() sets the name of icon window:

```
        static char[] = "Icon";
WindSet( win, WF_ICONTITLE, icon_title);
```

If the icon title is not set, the same title is used for both normal and icon windows.

Programming guideline of WinDom

The standard functions

WindCreate() attributes the standard functions std_icn(), std_unicn(), std_allicn() to the messages **WM_ICONIFY**, **WM_UNICONIFY** and **WM_ALLICONIFY**.

std_icn()

The function calls directly WindSet() with right parameters (**WF_ICONIFY** mode). When a window is iconified the **WS_ICONIFY** flag of the *win->status* variable is set to 1.

std_allicn()

All windows are closed except the window targetted by the message. Each window has the **WS_ALLICNF** status. The targetted window is iconified with the function WindSet() and the desktop menu (if defined) is disabled with the function MenuDisable().

std_unicn()

This function uniconifies the targetted window. The WS_ICONIFY status is unset. If the window has the WS_ALLICNF status, all windows closed with the WS_ALLICNF status are opened at their previous location. The desktop menu (if defined) is restored using MenuEnable().

Programming guideline of WinDom

Drawing the icon windows

When a window is iconified, the window workspace should have a special apparence (usually an icon). An iconified window is still a window so EvntWindom() still refreshes it using the same redraw function. To customize that, there are two ways:

- Inside the redraw function, you test if the window is iconified with the **WS_ICONIFY** flag of the *win->flag* variable and you adapt the display.
- If the window uses a predefined redraw function (like formulars for instance), the previous method is bad because you have to code a new redraw function. To overcome this problem, Windom uses a specific redraw function when the window is iconified and you can call WindSet() with the **WF_ICONDRAW** mode to define this specific redraw function. Syntax is :

  ```
  WindSet( win, WF_ICONDRAW, icon_draw);
  ```

  The icon redraw function has the same prototype than a standard window redraw function and obeys to the same rules.

Programming guideline of WinDom

Icon title

When a window is iconified, the same title than the uniconified window is used. As the icon window is usually small, it could be interesting to give a new title to this icon, generally a short one. It is possible with WindSet() and the mode **WF_ICONTITLE**. This defines the title of the window when it is iconified. This call can be performed at any time, even when the window is iconified (as **WF_NAME**). Example:

```
static char win_title[] = "My long window title";
static char icon_title[] = "ICON";

/* Define the window title when it is uniconified
 * If the window is uniconified when proceeding this
 * call, the window gets the new title */
WindSet( win, WF_NAME, win_title);
/* Define the window title when it is iconified
 * If the window is iconified when proceeding this
 * call, the window gets the new title */
WindSet( win, WF_ICONTITLE, icon_title);
```

Programming guideline of WinDom

Window dialog boxes

WinDom provides functions to handle very easily dialog boxes (also called forms) in windows. WinDom supports two types of them:

modeless forms
: that are seen like normal windows and are handled exactly like windows.

modal forms
: that use a modal window. Your application is stopped until the dialog box releases control but the AES and the other applications are still running if your system supports multitasking. The form is handled exactly like old classical GEM dialog boxes.

Programming guideline of WinDom

Modeless forms

There are two steps:

- creating the form,
- analyzing the result of the user interaction with the form. Therefore, this step is optional.

1 Creating the dialog box

To create it, use FormCreate():

```
WINDOW *FormCreate( OBJECT *tree, int Widget, VOID *proc,
                                        GRECT *coord, int weffect, int dup);
```

It is a huge function. As you can see, you have to give the address of an AES object tree (the form itself). First, the function verifies if the form is already created i.e. if a window form owns the same object tree. In this case the window is either reopened, uniconified or topped in foreground. It is possible to create several forms with the same object tree with the option *dup* set to 1. In this case the object tree is duplicated in memory (An object tree can be manually dupicated with the ObjcDup() function, the **WS_FORMDUP** flag of the window status - i.e. the *win->status* variable - should be set to 1 meanning that the duplicated object tree memory must be released by the window destroy function). Second, FormCreate() attaches standard event functions to the created window. These standard functions are devoted to form managing and seting some critical window variables.

2 form managing

Now the form is created and is displayed by your application. You have to analyze the result of the user interaction with your dialog box objects.

Example 1: using the main loop event

Here is a typical example, where we create a form (after a desktop menu selection), and then analyze the result:

```
/* The main loop event */
while(1) {
  EvntWindom( MU_MESAG);

  if( evnt.buff[0] == MN_SELECTED) {
       /* one selects an item int the desktop menu */
    switch( evnt.buff[4]) {
    case MENU_ITEM_FORM1:
            /* Create the form */
      rsrc_gaddr( 0, FORM1, &tree);
      FormCreate( tree,                  /* the object tree */
                                  MOVER|NAME|SMALLER,    /* window widgets */
                  NULL,                  /* no function result */
                  "Window title",
                  NULL,     /* The position of the form, if NULL,
                                                * the form will be centered, and the window size
                                                * computing according to the object root size */
                  1,        /* activate the graf_growbox effects */
                  0);       /* Don't duplicaterd the object tree */
      break;
      ....
    }
  }
  if( evnt.buff[0] == WM_FORM) {
             /* this message means that the user has clicked an EXIT
              * or TOUCHEXIT object in a form */
             WINDOW *form;
             int obj;

             form = WindHandle( evnt.buff[3]);       /* Get the window descriptor of the form */
             obj = evnt.buff[4];             /* index of the targetted object */
             switch ( objc) {
             case BUT_OK:
                     break;
             ...
             }
             ObjcChange( OC_FORM, win, objc, NORMAL, 1);     /* set the object state to NORMAL */
  }
}
```

Example 2: using an event function

The second part of the previous example can be placed in an event function. The *proc* parameter of the FormCreate() function defines this function. When EvntWindom() receives an WM_FORM message, its executes directly this function. The event function have the

folowing structure:

```
void DoForm( WINDOW *win ) {
  int objc = evnt.buff[4];

  switch( objc) {
    case BUT_OK1:
      ....
      break;
      ....
  }
  ObjcChange( OC_FORM, win, objc, NORMAL, 1);
}
```

and the FormCreate() call is:

```
        FormCreate( MOVER|NAME|SMALLER, tree, DoForm, "Window title", NULL, 1, 0);
```

ObjcChange() is the WinDom equivalent of the AES objc_change() function.

Remark for expert:

If you want to create yourself a form without the FormCreate() function, we list the main important steps to respect:

1. Create the window with the WindCreate() function,
2. Use the FormAttach() function which attributes a form to a window.
3. other initializations (name, menu, toolbar, etc)
4. use the GrectCenter() and WindCalc() functions to compute the window size.
5. open the window (with WindOpen()).

Programming guideline of WinDom

The modal form

With the predefined modal form, the point of view is completly different: forms are handled in a similary way than the classical GEM forms (see the GEM form library). The idea is that a modal form stops the program (like the classical forms), but not the other applications. So, there are three steps:

1. Display the form: function FormWindBegin(),
2. Handle the events: function FormWindDo(),
3. Close the form: function FormWindEnd().

Let's see in details these three functions.

```
WINDOW *FormWindBegin( OBJECT *form, (!B)char(!b) *nom);
```

creates the form window centered on screen (using the function GrectCenter() internaly) and returns the window descriptor. The default widgets of the window are a moving bar, a title and a smaller button but these widget can be changed by the user via the configuration file. The object tree *form* should containt a EXIT or TOUCHEXIT object.

```
(!B)int(!b) FormWindDo( (!B)int(!b) evnt);
```

this function returns the index of the last object selected. The parameter *evnt* sets the event to handle: the **MU_MESAG** evnt should be handled. We will see that all events can be handled if needed.

```
void FormWindEnd( void);
```

the function close the windom.

First example:

In this example, we draw a modal form and we return the index of the selected object.

```
int CallDialog( int index) {
   OBJECT *dialog;
   int res;

   rsrc_gaddr( 0, index, &dialog);
   FormWindBegin( dialog, "Formulaire");
   res = FormWindDo();
   FormWindEnd();
   return res;
}
```

Notice that, si the user clicks a EXIT or TOUCHEXIT object, the function terminates. It is not possible to handled, for example, a slider in this example.

Second example:

```
int CallDialog( int index) {
   OBJECT *dialog;
   int res;

   rsrc_gaddr( 0, index, &dialog);
   FormWindBegin( dialog, "Formulaire");
   do {
      res = FormWindDo( MU_MESAG);
      switch( res) {
         case OBJ_TOUCHEXIT1:
            ....
            break;
         ....
      }
   } while( dialog[res].ob_state & TOUCHEXIT);
   FormWindEnd();
   return res;
}
```

In this example, only EXIT objects selected can terminate the loop.

Third example:

The FormWindDo() function can handled all GEM event. If you set to 1 the bit **FORM_EVNT** of the *evnt* parameter of FormWindDo(), the function returns the last event detected by EvntWindom() (use internaly by FormWindDo()). For example, the call:

```
               res = FormWindDo( MU_MESAG|MU_TIMER|FORM_EVNT);


return the value MU_TIMER|FORM_EVNT if a timer
event occurs otherwise it returns an object index.


     int quit = 0;

     rsrc_gaddr( 0, index, &dialog);
     FormWindOpen( dialog, "Formulaire");
     do { /*
                * loop on AES events (bit FORM_EVNT)
          */
          res = FormWindDo( MU_MESAG|FORM_EVNT);
          if( res & FORM_EVNT) {  /* A AES event occurs
                                                                * (in this case, only MU_MESAG
                                                                * is possible */

               if( res & MU_MESAG && evnt.buff[0] == AP_TERM) {
                    snd_msg( NULL, AP_TERM, 0, 0, 0, 0);
                    quit = 1;
               }
          } else { /* handle the form ... */
               switch( res) {
               case OK:
                    quit = 1;
                    break;
                    ...
               }
          }
     } while( !quit );
     FormWindEnd();


Fourth example:


From Windom version of November 1999, it is possible to handle all
AES evnt using the fonctions from the Evnt Library. In particular,
FormWindDo() uses the EvntWindom() function. So to trap an AES event
during a FormWindDo() call, just use the EvntAttach() function.



{
          void *oldfunc;
     int quit = 0;

     rsrc_gaddr( 0, index, &dialog);
     FormWindOpen( dialog, "Formulaire");
          /* backup the old value */
          oldfunc = Evntfind( NULL, AP_TERM);
          /* give a new value */
     EvntAttach( NULL, AP_TERM, local_apterm);
     /* handle the form ... */
          do {
                  res = FormWindDo( MU_MESAG);
                  switch( res) {
          case OK:
                  quit = 1;
                  break;
              ...
          }
          } while( !quit);
          /* Restore value */
          EvntAttach( NULL, AP_TERM, oldfunc);
     FormWindEnd();
}


Important remark : the functions binded by EvntAttachj() have a
```

global action. So it is possible sonme function should be used during a FormWindDo() or any function using EvntWIndom(). In this case, the function should be provisary discarded (using EvntDelete()).

Programming guideline of WinDom

Binded objects

From the version fev 2000 of WinDom, there is a new - a third - way to handle forms in windows: it is possible to bind an object of a form to a variable or a function with the ObjcAttach() function.

Bind to a variable :

The idea is to link a selectable object to a variable: when the user selects or unselects this object, the variable binded is automatically updated. There are two cases :

- radio and selectable objects : the radio buttons of a form should be linked to a same integer variable. This variable contains the index number of the selected radio button. Example :

```
int radio_choice = RADIO1;      /* a global variable */

ObjcAttach( OC_FORM, win, RADIO1, BIND_VAR, & radio_choice, 0);
ObjcAttach( OC_FORM, win, RADIO2, BIND_VAR, & radio_choice, 0);
ObjcAttach( OC_FORM, win, RADIO3, BIND_VAR, & radio_choice, 0);
ObjcAttach( OC_FORM, win, RADIO4, BIND_VAR, & radio_choice, 0);
```

- non-radio and selectable objets : an integer variable linked to this object contains at any time the value of the **SELECTED** bit of the *ob_state* field of the object. Currently, only the **SELECTED** state may be binded.

```
int but1_state = 0;     /* 1 if the BUT1 object is selected */

ObjcAttach( OC_FORM, win, BUT1, BIND_VAR, & but1_state, 0);
```

  It is possible to attribute a variable to a specific bit (because bit field are often used to represent several options) :

```
int options = 0x1;      /* means the BUT1 object is selected */

/* Attach to bit 0 of options */
ObjcAttach( OC_FORM, win, BUT1, BIND_BIT, & options, 0x1);
/* Attach to bit 1 of options */
ObjcAttach( OC_FORM, win, BUT2, BIND_BIT, & options, 0x2);
```

Bind to a function:

If the simple SELECTABLE objects represent the value of a variable, the EXIT or TOUCHEXIT objects often require a complex operation. For example, the OK button closes the form or the SAVE button save the parameters. For that purpose, these objects may be linked to a function. If the user selects these objects, the binded function is invoked. Thus, it is not neccessary to write a **WM_FORM** event function to handle a form. Example :

```
   /* This function unselected the selected objet and
* close the window. A binded object function has the
* following prototype:
* void Func( WINDOW *win,          // window descriptor
*            int   index,          // index of the selected object
*            int    type);         // OC_FORM or OC_TOOLBAR
*/
```

```
void OkBut( WINDOW *win, int index, int type) {
        ObjcChange( type, win, index, NORMAL, 1);
        ApplWrite( app.id, WM_CLOSED, win->handle);
}

ObjcAttach( OC_FORM, win, OK_BUT, BIND_FUNC, OkBut);
```

Programming guideline of WinDom

Menus

In this section we explained how use desktop menu and menu in window.

Programming guideline of WinDom

Declare the menu

Window menu:

A window menu is attributed to a window using the WindSet() function (as AES philosophy). First of all, create the window with the WindCreate() function and get the menu address. Then use the WindSet() function with **WF_MENU** option.

Here a typical example :

```
{
     OBJECT *menu;
     WINDOW *win;

     win = WindCreate( NAME|MOVER|CLOSER, app.x, app.y, app.w, app.h);
     rsrc_gaddr( 0, MENU1, &tree);
     WindSet( win, WF_MENU, tree, NULL);
     WindOpen( win, -1, -1, 400, 200);
}
```

A menu can be removed by the call :

```
     WindSet( win, WF_MENU, NULL);
```

All menus attributed to a window are duplicated in memory (WindSet() uses the ObjcDup() function).

Desktop menu

The desktop menu is declared with the function MenuBar(). A typical example is :

```
main() {
     OBJECT *menu;

     ApplInit();
     RsrcLoad( "resource.rsc");
     rsrc_gaddr( 0, MYMENU, &menu);
     MenuBar( menu, 1);

     ...
}
```

Programming guideline of WinDom

Handle the menu

There are three ways :

- handle directly the **WM_MNSELECTED** message (for window menu) or the **MN_SELECTED** message (for desktop menu),
- use a function binded to **WM_MNSELECTED** message or **MN_SELECTED** message,
- binded each object in the menu.

We describe the three way to proceed.

handle WM_MNSELECTED

This message is returned by EvntWIndom() when a window menu item is selected by the user. The message has the following structure :

```
evnt.buff[0] = WM_MNSELECTED
evnt.buff[1] = application identifier
evnt.buff[2] = always 0
evnt.buff[3] = window targetted handle
evnt.buff[4] = title selected index
evnt.buff[5] = item selected index
```

Here a typical example :

```
main() {
    int res, title;
    WINDOW *win;

    /* create a window with a menu */
    ...
    while( 1) {
        res = EvntWindom( MU_MESAG);
        if( res & MU_MESAG && evnt.buff[0] == WM_MNSELECTED) {
            title = evnt.buff[4];
            /* get the window targetted */
            win = WindHandle( evnt.buff[3]);
            /* Handle the menu */
            switch( evnt.buff[5]) {
            case ITEM1:
                break;
            case ITEM2:
                break;
            }
            /* unhighlight the menu title */
            MenuTnormal( win, evnt.buff[4], 1);
        }
    }
}
```

handle MN_SELECTED

This message, returned by EvntWindom() when a user selects a desktop menu item, has a similar structure than the previous one :

```
evnt.buff[0] = WM_MNSELECTED
evnt.buff[1] = application identifier
evnt.buff[2] = always 0
evnt.buff[3] = title selected index
evnt.buff[4] = item selected index
```

And now an example :

```
main() {
    int res, title;
    WINDOW *win;

    /* Diverses WinDom initialization ... */

    while( 1) {
        res = EvntWindom( MU_MESAG);
        if( res & MU_MESAG && evnt.buff[0] == MN_SELECTED) {
            title = evnt.buff[3];
            /* Handle the menu */
            switch( evnt.buff[4]) {
            case ITEM1:
                break;
            case ITEM2:
                break;
            }
            /* unhighlight the menu title */
            MenuTnormal( NULL, evnt.buff[3], 1);
        }
    }
}
```

Bind event WM_MNSELECTED to a function

A function can be linked to the window menu event with the call :

```
WindSet( win, menu, do_menu);
```

where *menu* is the address of a menu object tree and *do_menu* is an event function which handle the user selection in the menu. This function is called by EvntWindom() when event **WM_MNSELECTED** occurs. This method have the advantage to have a global action in WinDom : each time EvntWindom() function is invoked, user selection of menu is taken into account.

```
void main()
{
    int res;
    WINDOW *win;  /* target window */
    OBJECT *tree;
    void do_menu( WINDOW *);

    ApplInit();
    RsrcLoad( "menu.rsc");
    /* create the window */
    win = WindCreate( NAME|MOVBER|CLOSER|SMALLER, app.x, app.y, app.w, app.h);

    /* add a menu and an event menu function */
    rsrc_gaddr( 0, MY_MENU, &menu);
    WindSet( win, WF_MENU, menu, do_menu);

    while(1) EvntWindom( MU_MESAG);

    RsrcFree();
    ApplExit();
}

/* Here the menu function */

void do_menu( WINDOW *win) {
    int title = evnt.buff[4];

    switch(( evnt.buff[5]) {
    case ENTRE1:
         break;
    case ENTRE2:
         break;
    }
    MenuTnormal( win, title, 1);
 }
```

Note : the call

```
WindSet( win, WF_MENU, menu, do_menu);
```

is strictly equivalent to :

```
    WindSet( win, WF_MENU, menu, NULL);
    EvntAttach( win, WM_MNSELECTED, do_menu);
```

Bind event MN_SELECTED to a function

As **WM_MNSELECTED** message, **MN_SELECTED** message, which designs user selection in the desktop menu, can be binded to a function. It is performed by the following call :

```
    EvntAttach( NULL, MN_SELECTED, do_menu);
```

Bind menu object to function

It is the third method to handle a menu in WinDom. Instead of handle the user selection event by catching an event message and trait it in a switch structure, we can attach each entry of a menu to a specific function. The method is the same for window menu and desktop menu. The following call :

```
    ObjcAttach( OC_MENU, win, QUIT, BIND_FUNCTION, quit);
```

attached the quit() function to the entry **QUIT** of the menu of the window *win*. If win is NULL, the desktop menu is addressed.

Object selection event function have a different prototype than event function :

```
    void quit( WINDOW *win, int obj, int mode, int title);
```

where *win* is the window descriptor containing the menu (it is **NULL** for the desktop menu, *obj* is the index of the selected entry, *mode* is equal to **OC_MENU** and *title* is the index of menu title hilighted by the user selection. This value is typically used by MenuTnormal() to unhilght the menu title. Example :

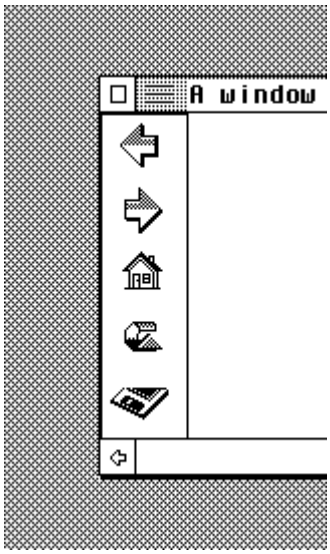
```
#include <windom.h>
#include <stdlib.h>

/* Quit the application */

void quitapp(void) {
    while( wglb.first) {
        ApplWrite( app.id, WM_DESTROY, wglb.first->handle);
        EvntWindom( MU_MESAG);
    }
    RsrcFree();
    ApplExit();
    exit( 0);
}

/* Give information */

void information( WINDOW *win, int obj, int mode, int title) {
    FormAlert( 1, "[1][We have selected item %d][OK]", obj);
    MenuTnormal( NULL, title, 1);
}

/* Open a window */

void openwin( WINDOW *win, int obj, int mode, int title) {
    WINDOW *win;
    OBJECT *menu;

    win = WindCreate( WAT_NOINFO, app.x, app.y, app.w, app.h);
    rsrc_gaddr( 0, WINMENU, &menu);
    WindSet( win, WF_MENU, menu, NULL);
    ObjcAttach( OC_MENU, win, CLOSE, BIND_FUNC, closewin);

    WindOpen( win, -1, -1, 300, 200);
    MenuTnormal( win, title, 1);
}

/* Close the window */

void closewin( WINDOW *win, int obj, int mode, int title) {
    MenuTnormal( win, title, 1);
}

void main( void) {
    OBJECT *menu;
```

```
    WINDOW *win;

    ApplInit();
    RsrcLoad( "resource.rsc");

    rsrc_gaddr( 0, DESKMENU, &menu);
    MenuBar( menu, 1);
    ObjcAttach( OC_MENU, NULL, QUIT, BIND_FUNC, quitapp);
    ObjcAttach( OC_MENU, NULL, INFORMATION, BIND_FUNC, information);
    ObjcAttach( OC_MENU, NULL, OPEN, BIND_FUNC, openwin);

    for(;;) EvntWindom( MU_MESAG);
}
```

Programming guideline of WinDom

Toolbars

A toolbar is a formular displayed in a part of a window. A toolbar can be vertical or horizontal. If the root object has his width bigger than his height, it will be displayed horizontally at the top of the window else it will be displayed vertically on the left of the window (see the following figures).



Put a toolbar in a window
Handle a toolbar

Programming guideline of WinDom

Put a toolbar in a window

Just use the WindSet() function with the **WF_TOOLBAR** mode. The call :

```
WindSet( win, WF_TOOLBAR, tool, do_tool);
```

puts the toolbar *tool* in the window *win*. All toolbar's events with be catch by the function *do_tool*. The parameter *do_tool* can be NULL. The call :

```
WindSet( win, WF_TOOLBAR, NULL, NULL);
```

removes the toolbar from the window *win*.

Programming guideline of WinDom

Handle a toolbar

As the window menus, there are two ways to handle a toolbar in a window :

- handle directly the **WM_TOOLBAR** message,
- use a function linked to the toolbar's events.

Direct method

AES 4.1 features toolbars utilities. However WinDom uses its own internal toolbar functions. For compatibility, WinDom uses the same AES 4.1 toolbar message when an object from a toolbar is selected. The **WM_TOOLBAR** message has the following structure :

```
evnt.buff[0] = WM_TOOLBAR
evnt.buff[1] = AES application identifier
evnt.buff[2] = always 0
evnt.buff[3] = window handle
evnt.buff[4] = selected object index
evnt.buff[5] = keyboard state
```

Event function

The event toolbar function is used exactly like the event menu function or the event formular function. Here, a typical example :

```
int main( void) {
    int res;
    WINDOW *win;
    OBJECT *tool;
    void do_menu( WINDOW *);

    /* create the window
         ...
     */

    /* Insert the toolbar and attach the do_toolbar
     * function to toolbar events */
    WindSet( win, WF_TOOLBAR, tool, do_toolbar);

    while(1)
        res = EvntWindom( MU_MESAG);
 }

/* This function handles the toolbar events */
void do_toolbar( WINDOW *win) {
    switch( evnt.buff[5]) {
    case BUTTON1:
         break;
    case BUTTON2:
         break;
   }
    ObjcChange( win, evnt.buff[4], 1);
}
```

Object binded

Off course, as dialog box or menu, it is possible to binded object from a toolbar to a function (or a variable) with the ObjcAttach() function.

Programming guideline of WinDom

Extended types for objects

Programming guideline of WinDom

Userdef objects and extended types

Objects are caracterized by their type : the *ob_type* field of the OBJECT **structure**. For example, the buttons have a **G_BUTTON** type. AES includes some predefined type and it is possible to create a new type. For that purpose, a special type is used : the **G_USERDEF**. This type means that the aes call a special function when it has to draw the object. A **G_USERDEF** object has an *ob_spec* pointing to a special structure containing the address of the drawing object function. To identifiate an object, AES uses only the low byte of the *ob_type* field. So, the high byte value can be used by the user. In particular, we use this field to give a second type to the userdef objects. This value is called the extended type. All ressource editors offer the possibility to edit this value.

Programming guideline of WinDom

MyDial compatibility

From the begining of AtariST, many high level GEM programming libraries proposed their own custom objects using the userdef objects. The most famous library is perhaps the MyDial library. The ressource editor Interface uses MyDial to display news objects and it is really cool to see in the ressource editor what you'll see in your program. For that purpose, WinDom provides a set of predefined objects compatible with MyDial. MyDial can even be used instead of WinDom to display custom objects.

Programming guideline of WinDom

Extended ressources

To attribute an extended object, WinDom (actually the RsrcXtype() function) scans the objects inside a ressource (or an integrated ressource) and examines the value of *ob_type* :

- the low byte of this value is **G_USERDEF**, RsrcXtype() does nothing, because it could be a special object reserved by the developper,
- the high byte of this value is not null, it is an extended type. RsrcXtype() gives a new type to the object (the **G_USERDEF**) the real. Depending the extended type, RsrcXtype() attributes to the *ob_spec* field a structure USERBLK containing the right userdef function (this function will be called by AES to draw the object) and the real object complet type (then changes done by RsrcXtype are reversible).

Programming guideline of WinDom

Programming with extended objects

There is no difference to use a normal obect or an extended one. If the extended object has a text, you have to use the ObjcString() function to read or change this text because the access is different with these objects.

To install the extended objects, you have to call the RsrcXtype() function. There are two case :

1. The ressource is extern (load in memory with the RsrcLoad(), not the rsrc_load() function!), you just have to invoke :

   ```
   RsrcXtype( 1, NULL, 0);
   ```

   This call installs the extended objects,

2. the ressource is intern (it is included in the source during, the compilation), the color icons have to be fixed to the current screen resolution with the function RsrcFixCicon() then the extended object are installed with RsrcXtype().

We illustrate these two case with a C example.

First case : external ressource

```
#include <windom.h>
#include "myrsc.h"

void main(void) {
    ApplInit();

    /* WinDom version of rsrc_load() */
    RsrcLoad( "myrsc.rsc");

    /* Extended type are installed : */
    RsrcXtype( 1,       /* Install the new types */
               NULL,    /* Work on the external ressource */
               0);      /* idem */

    /* body of program */

    /* End of program */

    /* Uninstall the extended objects */
    RsrcXtype( 0, NULL, 0);

    /* Free up the memory and AES */
    RsrcFree();
    ApplExit();
}
```

Second case : integrated ressource

```
#include <windom.h>
#include "myrsc.h"
#include "myrsc.rh"
#include "myrsc.rsh"
```

```
void main( void) {
    int dum;
    XRSRCFIX fix;    /* Used by RsrcFixCicon() and
                      * RsrcFreeCicon() */

    ApplInit();

    /* Fixe the oject coordinate to the screen resolution */
    for( dum=0; dum<NUM_OBS; dum++)
         rsrc_obfix( rs_object, dum);

    /* Install the extended objects */
    RsrcXtype( 1,           /* Install */
               rs_trindex,  /* address of tree objects */
               NUM_TREE);   /* Number of tree objects */

    /* Note : you can use simultaneously severals
              intern ressources and an extern ressource */

    /* If the ressource contains color icons, fix it to
     * the current screen resolution */
    RsrcFixCicon( rs_object, /* address of objects */
                  NUM_OBS,   /* number of objects */
                  NUM_CIB,   /* number of color icons */
                  NULL,      /* an optional color palet */
                  &fix);     /* Used later by RsrcFreeCicon() */

    /* body of program */

    /* Free up memory used by the color icons */
    RsrcFreeCicon( &fix);

    /* Free zup memory and AES */
    RsrcXtype( 0, rs_trindex, NUM_TREE);
    ApplExit();
}
```

Programming guideline of WinDom

Extended types and ressource editor

In the ressource editor, you can set the extended type of an object. The following table lists all extended provided by WinDom.

Avalaible extended type

~ means the state/value is forbidden
[] means an optional state/value
STATEn means the n bit of the *ob_state* field
DRAW3D is an alias of STATE7 (used by Interface)

Note : the first bit has a 0 index.

Extended types provided by WinDom

Nom	Description	ob_type	type	ob_flag	ob_state	Shortcut
XTEDINFO	Long boxed text	G_(BOX)TEXT	11			No
XBOXLONGTEXT	Long boxed text	G_(BOX)TEXT	12			No
XEDIT	Editable field	G_(F)TEXT	14	EDITABLE	[DRAW3D]	No
MENUTITLE	Menu item	G_STRING	15			No
MENUTITLE	Menu title	G_TITLE	15			No
ONGLET	Thumb index	G_BUTTON	16	RBUTTON	[DRAW3D]	Yes
	button	or G_STRING			[STATE8]	
					[STATE15]	
ONGLET	Thumb index	G_BUTTON	16	~ RBUTTON	[DRAW3D]	Yes
	background	or G_STRING				
DIALMOVER	Background form	G_BOX	17		[DRAW3D]	No
					[OUTLINE]	
DCRBUTTON	Radio button	G_BUTTON	18	RBUTTON	[DRAW3D]	Yes
		or G_STRING			[STATE8]	
DCRBUTTON	Checked button	G_BUTTON	18	RBUTTON	[DRAW3D]	Yes
		or G_STRING		EXIT	[STATE8]	
DCRBUTTON	Exit button	G_BUTTON	18	EXIT	[DRAW3D]	Yes
		or G_STRING		RBUTTON		
				[DEFAULT]		

CIRCLEBUT	Cycle button	G_BUTTON	22		[DRAW3D]	No
		or G_STRING				
UNDERLINE	Underlined texte	G_BUTTON	19		[DRAW3D]	Yes
		or G_STRING				
TITLEBOX	Frame with title	G_BUTTON	20		[DRAW3D]	No
HELPBUT	Help button	G_BUTTON	21			No
	Text with	G_BUTTON	24		[DISABLED]	Yes
KPOPUPSTRG	keyboard shortcut	or G_STRING				
SLIDEPART	Box Char	G_BOXCHAR	25		[DRAW3D]	No
UNDOHELP	Undo button	G_BUTTON	31			No
-	Undo object	-	-	FLAG11		No
-	Relief button	-	-		[DRAW3D]	No
USERDRAW	Userdraw object	-	255			No

SLIDEPART is an object used to create boxchar (specially for scrolling objects) having the same apparence under TOS, Naes or MagiC.

Windom specific objects

- XTEDINFO is used to display G_TEXT and G_BOXTEXT object with the same aspect (in DRAW3D mode) on TOS, MagiC or any AES replacement.
- XBOXLONGTEXT is a BOXTEXT object displaying the text on several lines if needed. ObjcEdit() is used as interface to the object.
- XEDIT is an editable objet with an unlimited (except the memory) text length. It is very recommanded to use it rather than classic EDITABLE object. ObjcEdit() is used as interface to the object.
- MENUTITLE is used as menu title and menu item object. The text can be display using a specific font (see windom.menu.font). Keyboard shortcut are correctly aligned on the right even with a proportional font.
- ONGLET is a special object for multiple formular thumb indexed. As all extended object, it is sensible to the DRAW3D state.
- USERDRAW is a special type reserved by RsrcUserDraw. Please, never use this value for your own extended objects.

Mydial objects unsupported

The following objects are currently ignored by WinDom :

POPUPSTRG (23)
LONGINPUT (26)

and the bit 15 of ob_flags for editable object (multiple line editable field).

Usual extended states

Follow, the signification of the *ob_state* bits unused by AES but used by WinDom.

- DRAW3D draws the object with a relief effect,
- STATE8 draws the object with an alternative look :
    - radio button:
      STATE8: button have an MagiC look,
      ~STATE8: button have a xv look,
    - check button:
      STATE8: a square button with a white foreground color and a cross inside,
      ~STATE8: a square button with a ckeck symbol (OpenLook style),
    - thumb index:
      STATE8: thumb index are displayed with rounded corners,
      ~STATE8: thumb index are displayed with squarred corners,
    - underlined text:
      STATE8: the object is underlined,
      ~STATE8: only the text inside the object is underlined,
- All extended object including text support the following text attributs :

Attributs de texte

ob_state bit	Attribut
STATE9	**Bold**
STATE10	Underline
STATE11	*Italic*
STATE12	Outline
STATE13	Shadow
STATE14	Light

Note: the light attribut is used when an object has the state DISABLED.

Keyboard shortcuts

Some objects can display a keybord shortcut (see table ...). A keyboard shortcut is a underlined letter inside the text. It means that the object can be selected by typing the key combinaison [Alternate + letter]. To make appear this shortcut, just add a '[' character behing the letter. For example:

[Cancel -> Cancel -> alt+c.

Programming guideline of WinDom

Programming thumb indexes

WinDom provides extended objects to create multiple formular (ONGLET), AND a function to handle automatically these objects (see FormThumb() manuel which contains an example).

WinDom doesnot handle more than one multiple formulars per window dialog box. In these case, you have to handle explicitly.

It is really easy to handle that. We provide the algorithm. First of all, the algorithm provided supposes severals hypothesis :

1. Thumb index are placed under or upper the formulars. Only a single line of thumb index is possible.
2. Thumb index have to touch the formulars frames.
3. Thumb index are G_BUTTON with an extended type 16 and a RADIO flag. Formulars are G_BUTTON with an extended type 16 and are not RADIO.
4. The index of Thumb object have to ... Les index des boutons d'onglets doivent se suivre.
5. The formulars have to have the HIDDEN flag.

Constrains 1, 2 and 3 come from the way the objects (ONGLET) are drawn. Constrains 4 and 5 come from the algorithm used. The principle is very simple: the active formular is displayed and the others are hidden using the **HIDDEN** flag.

```
(!U)Example(!u)

/* Handle a multiple formular */

void formONGLET( WINDOW *win) {
     static int show = FORM1;   /* formular currently displayed */
     int bckgrd[] = {FORM1, FORM2, FORM3 /*, etc ...*/};
                                        /* Describes links between thumb index and
                                  * formular */
     int res = evnt.buff[4];

     switch( res) {
     /* The multiple formular handling is here */
```

```
      case INDEX1:
      case INDEX2:
      case INDEX3:
      /* ... */
                  /* Test if the choice is already displayed */
            if( show == bckgrd[res-BUT1]) break;
            /* Hide the current form */
            FORM(win)[show].ob_flags |= HIDETREE;
                  /* Unhide the new form */
            FORM(win)[bckgrd[res-BUT1]].ob_flags &= ~HIDETREE;
            /* keep in memory the form displayed */
            show = bckgrd[res-BUT1];
             /* this instruction fixes a bug from WinDom */
            ((W_FORM*)win->data)->edit = -1;
            /* Display the new form and the thumb index selected */
            ObjcDraw( OC_FORM, wglb.appfront, show, MAX_DEPTH);
            ObjcDraw( OC_FORM, wglb.appfront, res, 0);
            break;

      /* Others objects ... */
      case OK:
            ...
      }

}
```

Programming guideline of WinDom

Special text objects

WinDom defines two special text objects : XBOXLONGTEXT and XBOXLONGTEXT.

The first one allows you to display a long text in a BOXTEXT objet. The text can be display on sereval lines if needed. Just set the extended type in your ressource editor. Long text is read or written using the function ObjcString().

The second one is an editable object without limitation of text size. This object is very easy to handle and usefull and replaces effiencly the standard EDITABLE object. However, standard EDITABLE object can be used in a case of formated fields (such as date input for example) because XEDIT objects don't use the template string (as G_FTEXT objets).

Programming guideline of WinDom

The UserDraw objects

These objects are not really extended object. The extended type (255) should never be set directly by the user from the ressource editor. This value is not used by RsrcXtype() but by RsrcUserDraw().

The goal of these objects is to provide an easy way to draw something in a formular or a toolbar inside a window. Drawing inside classical GEM formular is not possible.

To attribute a drawing function to an object, you have to use the RsrcUserDraw() function. This function transforms the object in a special extended type (255) format. After this call, the AES will call the function given to RsrcUserDraw() to draw the object. The function given to RsrcUserDraw() - that we call the UserDraw function - and the Userdef function are differents. Actually, the Userdef function calls the drawing function to draw the object. The main raison of this system is that the drawing function is similar to a standard drawing function of a window (i.e. the function called by **WM_REDRAW**), but there are some differences:

- the UserDraw function have not the same argument. There is an additionnal parameter, a **PARMBLK** structure poviding all informations related to the object.
- we have the same limitations for the UserDraw function than the userdef function.

The only one difference with a Userdef function is that we should never clip the redraw area of the object. This action is performed by the userdef function calling your UserDraw function.

Example

/* A typical userdraw function : a simple text */

void MyUserDraw(WINDOW *win, PARMBLK *pblk) { char *p;

A FINIR }

Programming guideline of WinDom

Keyboard shortcuts

Programming guideline of WinDom

Keyboard shortcuts and WinDom

Keyboard shortcuts are handled directly by EvntWindom(). The developper does nothing except to declare the shortcuts. The declaration of these shortcuts are performed in the ressource. When a keyboard event occurs, EvntWindom() evaluates it and searchs among the formulars, toolbar and menus. When the shortcut is found, the search stops and a message is sent. The shortcut is successively search in:

1. the desktop menu (if found, a **MN_SELECTED** message is sent),
2. active window menu (if found, a **,WM_MNSELECTED** message is sent),
3. the active window toolbar (if found, a **WM_TOOLBAR** message is sent),
4. the active window formular (if found, a **WM_FORM** message is sent).

The active window may be the top window or the window pointed by the mouse (see the windom.evnt variable).

Programming guideline of WinDom

Keyboard shortcuts structure

Menus

Each keyboard shortcut appears in the menu as the last word of the item. The word must have a space character at the beginning and the end of the word. For example : é" Quitter ^Q ".

The keyboard shortcut can have the following first character (after the space character):

^ (0x5E) meanning Control
? (0x07) meanning Alternate
? (0x01) meanning Left Shift or Right Shift

These characters are optional.

The next character may a alphabetical character (a..z) or a special sequence representing a special key. These sequences are :

ESC	The Escape key
UNDO	the Undo key
HELP	the Help key
INSERT	the Insert key
HOME	the Home key
TAB	the Tab key
BACK	the Backspace key
DEL ! the Delete key	
F1 ... F10	the function keys 1, 2 ...

Examples

" Information I "
" Copy ^C "
" Center F1 "
" Help HELP "
" Delete ^DEL "
" Infos ? "

Formulars

Only the extended object 18 (button) can have a keyboard shortcut. If we insert a '[' character behind a character, this character will appear as underlined and the object will be selected by typing the sequence Alternate and the underlined character.

Example

text button: "[Save configuration"

appear as: "Sve configuration"
keyboard shortcut: alternate S

Others objects can be selected from the keyboard:

RETURN ou ENTER
selects the DEFAULT object,

UNDO
selects the object with a FLAGS11 ob_flags or the object with a 31-extended type,

HELP
selects the 21-extended type object.

Standard Editable fields (EDITABLE objects)

WinDom does not integrate high custom functions such as copy/paste. Nowadays, moderns AES (MagiC, Naes) integrates these functions. If you does not use MagiC or Naes, you can use Let's Them Fly, a TSR program compatible with all TOS versions. The functions offers by these programs are :

control right arrow, left arrow
jump to the next, previous word,

control up arrow, down arrow
acces the historic (only Let's Them Fly),

shift right arrow, left arrow
jump at the begining, the end of the field,

shift insert
displays an ascii table (only Let's Them Fly),

shift undo
recalls the previous field (only Let's Them Fly),

control C
copies in the GEM clipboard,

control V
paste the GEM clipboard,

escape
clear the field.

Extended Editable fields (XEDIT objects)

WinDom provides a special editable object (XEDIT) allowing to type a text without lenght limitation. Some special control keys can be used within these objects :

control right arrow, left arrow
jump to the next, previous word,

shift right arrow, left arrow
jump at the begining, the end of the field,

control C
copy in the GEM clipboard,

control V
paste the GEM clipboard,

escape
clear the field,

control K
kill the line at the cursor position.

Programming guideline of WinDom

Frame windows

Any window can be divide in several areas that we frame. Each of these frames are viewed like a standard window by WinDom. The main window containing the frame window have a specific status and, off course, specific event functions. A frame can be optionnally resized by the user. The more interesting thing is you can used any predefined window (by WinDom like formulars or user window) as frame and build complex window. A good example is a text editor using windows divided in several frames. Each frame is focused on a region of a same buffer.

Principle

A Frame window (i.e. the main window) is a standard window with specific event functions. The *data* field of the window point to a special structure describes the frames and other usefull informations. The event functions of the Frame window use the event function of the framd windows. For example, the redraw function draws the frames (borders of each areas) and calls the redraw function of the framed window on the correct aeras.

A framed window is always a WinDom window, described by a **WINDOW** structure but does not exist as a standard window. It exists only for the main window.

How create a frame

1. Initialize the frame environnement in WinDom: FrameInit(),
2. Create a main window wich will containt the frame: FrameCreate(),
3. Create the windows that we want include in the frame window (but don't open them),
4. Transform them in frame windows: FrameAttach(),
5. set some optional parameters: FrameSet(),
6. finaly, open the main window and handle the GEM events,
7. At the end, close the frame environnement: FrameExit().

The FrameFind(), FrameSearch() and FrameCalc() functions works specifically on a framed window. Others window function from the WinDom library can be used on framed window or standard window (for example the usefull WindSet()).

A lot of bugs occurs specially during the frame resizing events.

Programming guideline of WinDom

Fonts ...

Programming guideline of WinDom

The fontid file

The Font library offers some usefull functions to manipulate Fonts. These functions work when a font driver (such as Gdos, Speedo-Gdos, Ndvi or equivalent) is available. However, these calls - except vqt_xname() - work even if the font driver is not avalaible : in this case, WinDom reads a special file which describe fonts available when the font driver is not in memory. This file - **fontid** - is searched in the following paths:

- current application path,
- **$ETCDIR** path,
- **$HOME** path,
- **$HOME**\Defaults path,
- **$FONTDIR** path,
- C:\gemsys\ path.

The file has the following structure :

```
# @(#)WinDom/fontid
# Copyright Dominique Béréziat 2000
# Describe the font features when there is no font driver.

index "font name" font-id font-flags
```

The file can be generated automatically by the program fontid.ttp from the WDK package.

Programming guideline of WinDom

A small example

Programming guideline of WinDom

Event messages used by WinDom

We describe the WinDom specific event messages (MU_MESAG events) and their significations. Important: remember that EvntWindom() tries to executed the function attached to a message when an event message occurs.

WM_DESTROY

This message means the targeted window should be destroyed. WinDom makes the distinction between **WM_CLOSED** that means the window should be closed on the screen but stays in memory and this message that means :

- data attached to this window should be saave then destroy
- the window should be close and remove from the memory

Note: on singleTOS, this message is sent when a user clicks on the closer window widget with the shift key pressed.

evnt.buff[0] = WM_DESTROY
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle

WM_BOTTOMED

This message is standard from AES 4.0. It means that a window should be sent in the background (at the bottom of the window liste). This message is emulated by WinDom if the system does not support it. If the system does not support the bottomer widget, a window can be sent to background by shift clicking the widget mover bar.

evnt.buff[0] = WM_BOTTOMED
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle

WM_ICONIFY, WM_UNICONIFY, WM_ALLICONIFY

These message are standards from AES 4.1 . However, WinDom emulates them (if there are not available in the system) to handle the window iconification.

A window should be iconified ...

evnt.buff[0] = WM_ICONIFY
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle evnt.buff[4-7] = position and size of the icon window

A window should be uniconified ...

evnt.buff[0] = WM_UNICONIFY
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle evnt.buff[4-7] = position and size of the uniconified window

All windows should be iconified ...

evnt.buff[0] = WM_ALLICONIFY
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle evnt.buff[4-7] = position and size of the main icon window

WM_FORM

These message means that a selectable object is selected (with the mouse or the keyboard) in a window formular.

evnt.buff[0] = WM_FORM
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle
evnt.buff[4] = selected object index
evnt.buff[5] = keyboard state (see evnt_button())

WM_MNSELECTED

The message means a menu item is selected in a window.

evnt.buff[0] = WM_MNSELECTED
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle
evnt.buff[4] = title menu index
evnt.buff[5] = item menu index

WM_TOOLBAR

These message means that a selectable object is selected (with the mouse or the keyboard) in

a window toolbar.

evnt.buff[0] = WM_TOOLBAR
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = window handle
evnt.buff[4] = selected object index
evnt.buff[5] = keyboard state (see evnt_button())

AP_LOADCONF

When EvntWindom() received this message, the WinDom configuration file is reloaded. This message allows special WinDom application such as WinConf to parametrise in real time the Look'n Feel aspects of WinDom. It is a good idea to handle this message if you use the WinDom configuration file (see Conf library) to store your application parameters.

evnt.buff[0] = AP_LOADCONF
evnt.buff[1] = application id

AP_BUTTON

When EvntWindom() receives this message, a MU_BUTTON event is created. This message is used to simulated a MU_BUTTON event.

evnt.buff[0] = AP_BUTTON
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = coordinate x of the mouse
evnt.buff[4] = coordinate y of the mouse
evnt.buff[5] = mouse button state (see evnt_button())
evnt.buff[6] = keyboard state (see evnt_button())

AP_KEYBD

When EvntWindom() receives this message, a MU_KEYBD event is created. This message is used to simulated a MU_KEYBD event.

evnt.buff[0] = AP_KEYBD
evnt.buff[1] = application id
evnt.buff[2] = 0
evnt.buff[3] = scancode of the key hited
evnt.buff[4] = keyboard state (see evnt_button())

WM_UPLINED, WM_DNLINED, WM_UPPAGED, ...

These messages are strictly equivalent to WM_ARROWED messages (WA_UPLINED, WA_DNLINED, WA_UPPAGED). As these message are sub-mode of WM_ARROWED, they cannot be binded directly with EvntAttach(). It is now possible with the new messages WM_UPLINED ...

evnt.buff[0] = message
evnt.buff[1] = application id
evnt.buff[2] = 0

WM_PREREDRAW

This function attached to this message is called one and ony one time per WM_REDRAW event. A WM_REDRAW function can be called several time for a same event because because EvntWindom() calls the binded function for each rectangle of the AES rectangle list (it is the AES method to draw windows partially occlused). In some case, this WinDom feature can be an handicap, specially if you want perform one action per WM_REDRAW. In this case, you can catch the WM_PREREDRAW event which it call only one time (by EvntWindom()) for each WM_REDRAW event.

Notice, if you catch the WM_PREREDRAW event instead of the WM_REDRAW message, you can control complety the event and you disable the WinDom handling of redraw event (clipping on each AES rectangle).

Programming guideline of WinDom

Bubbles help (with BubbleGEM)

WinDom has function allowing you to call very easily BubbleGEM. BubbleGEM is a deamon provides to GEM applications bubbles help. The functions BubbleCall() and BubbleEvnt() display bubbles help on windows and the functions BubbleDo() and BubbleModal() display bubbles help inside a GEM formular. Use the BubbleAttach() function to bind a bubble help to an object in an objects tree. The BubbleConf() function configures locally the BubbleGEM behavior. However, BubbleGEM may be globally configured using the BubbleGEM CPX.

Programming guideline of WinDom

Some examples

Programming guideline of WinDom

BubbleGEM and the AV-protocol

When a bubble is drawn on the screen, BubbleGEM application takes the control of AES. If you click a mouse button, or if you hit a key the bubble disapears but as BubbleGEM has get the event, your application don't receive any event (**MU_BUTTON** event or **MU_KEYBD** event) but it could be very interesting the application receives these events (to make the application more reactiv from the user point of view). For that purpose, BubbleGEM sent to the application a message:

- a **AV_SENDCLICK** if the user clicked the mouse button when a bubble was displayed,
- a **AV_SENDKEY** if the user hited the keyboard when a bubble was displayed.

The application should react by transform these messages in **MU_BUTTON** event and **MU_KEYBD**. It can be done in WinDom by sending the **AP_BUTTON** and **AP_KEYBD** messages to the application.

Example:

```
/* Handle the AV_SENDKEY message */

void AvSendKey( void) {
        ApplWrite( app.id, AP_KEYBD, evnt.buff[3], evnt.buff[4]);
}

/* Handle the AV_SENDCLICK message */

void AvSendClick( void) {
        ApplWrite( app.id, AP_BUTTON, evnt.buff[3], evnt.buff[4]);
}

/* in the main part : declare the previous functions */

int main( void) {
        ...;

        EvntAttach( NULL, AV_SENDCLICK, AvSendClick);
        EvntAttach( NULL, AV_SENDKEY,   AvSendKey);

        ...;
}
```

Now, your application understands the AV_SENDKEY/BUTTON messages.

Programming guideline of WinDom

The AV protocol

Programming guideline of WinDom

What is the AV protocol ?

The AV protocol was introduced by the alternative desktop GEMINI. The idea was to use the GEM messages pipe to allows a custom communication between the desktop and the desktop accessories. With the new multitasking systems, this protocol was extended to any GEM applications. The AV protocol has a server : initialy the Venus application (the name of the Gemini Desktop) and clients : initialy the desktop accessories but now any GEM applications. Currently, only Thing the alternativ desktop handle completly the AV protocol but many desktop use it (Ease, MagxDesk, Neodesk, Jinnee). For more informations, read the hypertext documentation of Thing.

WinDom containts some usefull functions alllowing your applications to comunicate efficiently with the AV server and to use all the functionnalyties offer by the AV protocol.

Programming guideline of WinDom

Philosophy of the AV protocol

So the AV protocol is a set of AES messages exchange between the clients and the server. There are the messages sent from the serveur to the clients, they have the **VA_** suffix, and there are the message sent from a client to the server, they have the **AV_** suffix. Almost messages have an answer. For example, the **AV_STARTPROG** message, that is a request from a client to the server to exec an application, have an answer **VA_PROGSTART** meanning if the application was correctly launched or not. So there are mainly two king of messages:

two king of messages.

- a request, a message sent by the server to a client or sent by a client to the server to perform an action,
- an answer, that is a message anwser of a request

The requests are:

- protocol initialization message,
- action message.

Before to sent a request to a server, a client must be declared to the server that is performed by the AvInit() function. When the client finish, the AV session opened with AvInit() must be closed with the AvExit() function.

Send a message to the server:

The ApplWrite() and AvWaitfor() functions allow the client to send efficiently a message to the AV-server. The ApplWrite() just send a message and the AvWaitfor() waits for an answer form the server.

Handle messages from the AV server:

A client can declare to the AV server the actions that it can handle. In this case, the client can receive requests from the server that it have to handle.

Programming guideline of WinDom

Le protocol AV et EvntWindom()

The messages **AV_SENDCLICK** and **AV_SENDKEY** should be handled by your application specially if you use BubbleGEM (see section ...).

Programming guideline of WinDom

Diverses tables

Under construction ...

Client states (status parameter of AvInit())

Bit	Name	Signification
0	A_SETSTATUS	client supports the VA_SETSTATUS message
1	A_START	client supports the VA_START message
2	A_STARTED	client supports the AV_STARTED message
3	A_FONTCHANGED	client supports the VA_FONTCHANGED message
4	A_QUOTE	filename are quoted if needed
5	A_PATH_UPDATE	client supports the VA_PATH_UPDATE message

Server states (value returned by AvStatus())

Bit	Name	Word	Signification
0	V_SENDKEY	0	AV_SENDKEY message supported
1	V_ASKFILEFONT	0	AV_ASKFILEFONT message supported
2	V_ASKCONFONT	0	AV_ASKCONFONT message supported
3	V_ASKOBJECT	0	AV_ASKOBJECT message supported
4	V_OPENWIND	0	etc ...
5	V_STARTPROG	0	
6	V_ACCWINDOPEN	0	
7	V_STATUS	0	
8	V_COPY_DRAGGED	0	
9	V_PATH_UPDATE	0	
10	V_EXIT	0	
11	V_XWIND	0	
12	V_FONTCHANGED	0	
13	V_STARTED	0	
14	V_QUOTE	0	quoted filename supported
15	V_FILEINFO	0	
0	V_COPYFILE	1	
1	V_DELFILE	1	
2	V_VIEW	1	
3	V_SETWINDPOS	1	

For more information, see the hyptexte documentation of Thing about the AV protocol.

Programming guideline of WinDom

Gcc 32 bits portability

... or how write portable WinDom code ?

With the support of Gcc 32 bits, we have to take some good reflex to create a source file available for compiler which use integer ('int') with a size of 16 bits such as Pure C and Sozobon or compiler which use integer with a size of 32 bits. The first problem comes from GEM. GEM are coded with 16 bits integer and function binding use short integer. For that raison all GEM library for Gcc work with short integer (GemLib for example) or INT16 integer (MGemLib for example). INT16 is an 'int' for Pure C and a 'short' for Gcc. WinDom Functions addressed are mainly WindGet() and FrameGet().

So the first rule is :

use INT16 (defined by WinDom or MGemLib) when you use AES, VDI and WinDom functions. Look at the new WinDom specifications.

In the future, Pure C should use a modern GEM library as MGemLib and the naturel type used will be short integer that will be natural for each compiler.

The second problem comes with the way which parameters are transmitted to a function : in 32-bit mode, each parameter - even short integer - use a size of 32 bits. When you use functions, such as ApplWrite() and WindSet(), it is not possible to deal directly with pointer type due to the previous reason. Such as parameter should be encapsuled by a ADR() macro function (defined in WINDOM.H). In order to prevent these problems during compilation, prototype of WindSet() and ApplWrite() have changed for respectively 4 and 5 integer instead of variable arguments prototype if you use gcc with long integer. For other case, the old variable argument prototype has been kept.

So the second rule is :

With ApplWrite(), use ADR() macros function which pointer parameter. With WindSet() use WindSetStr() or WindSetPtr() macros for modes addressing pointer parameters (e.g. WF_TITLE, WF_MENU, ...).

Look at the DEMO program which compiles and wor$ks correctly for each compiler.

Programming guideline of WinDom

Configuration of WinDom applications

Programming guideline of WinDom

The philosophy

WinDom Configuration is performed via an unique file that we call the WinDom Configuration file. The main idea is: as WinDom is a GEM front end, the choice of window, object, ... apparence should not be fixed by the program but only by the user. For that purpose, WinDom reads an external file, the WinDom configuration file, the parameters of look and feel of the library.

In addition, WinDom offers to the programmer an easy way to store, read, and handle a configuration setup. Specific variable can be written and read from the windom configuration file.

The configuration is a text editable file. The syntaxe is very simple. However, a special application, WinConf written by the author, allows users to create and handle their configuration file. From WinConf version 2, any variables is handled (from WinDom or from specific application). The interaction is completly graphical (with buttons, popup menus, objects selectors). WinDom application can communicate with WinConf using a GEM protocol and the settings can be updated in real time.

Programming guideline of WinDom

The configuration file

Location of the configuration file

The configuration file is a file 'windom.cnf' or '.windomrc' if your file system supports the long file name. This file can be placed in the following directories :

- the application directory,
- the $HOME\Defaults directory,
- the $HOME directory,
- the $ETCDIR directory,
- the $PATH directories (can be multiple),
- the 'C:\' directory

The application directory allow to have a direct configuration of your application when a user install a first tilme your application.

The $HOME directories allow a personnal configuration in a multi user environnement.

The $ETCDIR directories allow WinDom appolication to live happily in an UniX-like file system organization. Perhaps it is better to read directly in the U:\ETC (or /etc) directory, i don't know, i'm waiting for users feedbacks.

Where applications are defined

First of all, commented lines begin by the '#' character. Example:

windom.version = true

The settings of an application are grouped in the same place between two special keywords: the application tag and the end tag. An application tag is just the application name (with higher characters) between brackets. Example:

```
[WINCONF] # begin of the configuration area of WINCONF application
# body of the application configuration
[end]     # end of configuration
```

Because a configuration addresses a specific application, it could be interesting to have a global settings addressing all applications. It is possible with the special tag [Default Settings]. An application not defined in the configuration file will read its settings in this area.

Variables

The syntax of a variable definition is really simple :

```
keyword = value [, value, ...]
```

Notice that a space character is required just behind the '=' character (it's a bug :(). The keyword is called too a variable.

Variables have a hierarchical structure. A name of variable is a list of group name, separated by a '.' character. Each group represents a thematic set of variables. Example, the variable

```
windom.event.keybd
```

is a 'windom' variable (i.e. a variable used by window) from the 'event' group of variables dedicated to handle the GEM events and it have the name 'keyboard' because it addresses the keyboard events. This orgazination is just a convention. By convention, the variables addressing the application begin by the application name :

```
myappli.window.save
```

From WinDom of May 05 1998, the WinDom variables have change their name. Few applications use this old version of WinDom. However WinConf is able to handle these old variables.

Programming guideline of WinDom

Hierarchical description of variables

Name

windom - WinDom configuration.

Type

Group

Subgroups

windom.evnt
windom.button
windom.exit
windom.string
windom.menu
windom.popup
windom.window
windom.fsel
windom.bubble
windom.mform
windom.iconify
windom.shortcut
windom.relief
windom.version

Description

All WinDom parameters are located in this group.

Programming guideline of WinDom

windom.evnt

Name

windom.evnt - WinDom Event configuration.

Type

Group

Subgroups

windom.evnt.button
windom.evnt.keybd

Description

This group configures WinDom behaviors with some AES events.

Parent group

windom

Programming guideline of WinDom

windom.evnt.button

Name

windom.evnt.button - WinDom Event Button configuration

Type

Variable

Syntax

windom.evnt.button = {mouse|front}

Default value

windom.evnt.button = front

Description

The variable sets the behavior of EvntWindom() with the MU_BUTTON event. The value mouse means that EvntWindom() applies the MU_BUTTON event to the window located under the mouse pointer (standard X11 behavior). The value front means that EvntWindom() applies the MU_BUTTON event to the window in the foreground (standard GEM behavior).

Parent group

windom.evnt

Programming guideline of WinDom

windom.evnt.keybd

Name

windom.evnt.keybd - WinDom Event Keyboard configuration

Type

Variable

Syntax

windom.evnt.keybd = {mouse|front}

Default value

windom.evnt.keybd = front

Description

The variable sets the behavior of EvntWindom() with the MU_KEYBD event. The value mouse means that EvntWindom() applies the MU_KEYBD event to the window located under the mouse pointer (standard X11 behavior). The value front means that EvntWindom() applies the MU_KEYBD event to the window in the foreground (standard GEM behavior).

Parent group

windom.evnt

Programming guideline of WinDom

windom.bubble

Name

windom.bubble - Bubble GEM configuration.

Type

Group

Subgroups

windom.bubble.font
windom.bubble.size

Description

This group configures the fonts used in bubble GEM.

Parent group

windom

Programming guideline of WinDom

windom.bubble.size

Name

windom.bubble.size - Bubble GEM text size

Type

Variable

Syntax

windom.bubble.size = Font size (in point)

Default value

windom.bubble.size = 13

Description

This variable sets the size of font used by bubble GEM.

Parent group

windom.bubble

Programming guideline of WinDom

windom.bubble.font

Name

windom.bubble.font - Bubble GEM font

Type

Variable

Syntax

windom.bubble.font = "Font name"

Default value

windom.bubble.font = "system font"

Description

This variable sets the font used by Bubble GEM.

Parent group

windom.bubble

Programming guideline of WinDom

windom.button

Name

windom.button - WinDom button configuration

Type

Group

Subgroups

windom.button.color
windom.button.font
windom.button.size

Description

This group configures the look of WinDom button (extended objects). Only no EXIT buttons are adressed.

Parent group

windom

Programming guideline of WinDom

windom.button.color

Name

windom.button.color - WinDom button text color

Type

Variable

Syntax

windom.button.color = AES colors index (0..15)

Default value

windom.button.color = 1 (BLACK)

Description

This variable sets the text color of simple buttons.

Parent group

windom.button

Programming guideline of WinDom

windom.button.size

Name

windom.button.size - WinDom button text size

Type

Variable

Syntax

windom.button.size = Font size (in point)

Default value

windom.button.size = 13

Description

This variable sets the text size of simple buttons.

Parent group

windom.button

Programming guideline of WinDom

windom.button.font

Name

windom.button.font - WinDom button text font

Type

Variable

Syntax

windom.button.font = "Font name"

Default value

windom.button.font = "system font"

Description

This variable sets the text font of simple buttons.

Parent group

windom.button

Programming guideline of WinDom

windom.string

Name

windom.string - WinDom string objects configuration

Type

Group

Subgroups

windom.string.color
windom.string.font
windom.string.size

Description

This group configures the look of WinDom simple object string: underlined text, boxtitle and popup label.

Parent group

windom.string

Programming guideline of WinDom

windom.string.color

Name

windom.string.color - WinDom string objects text color

Type

Variable

Syntax

windom.string.color = AES colors index (0..15)

Default value

windom.string.color = 1 (BLACK)

Description

This variable sets the text color of string objects.

Parent group

windom.string

Programming guideline of WinDom

windom.string.size

Name

windom.string.size - WinDom string objects text size

Type

Variable

Syntax

windom.string.size = Font size (in point)

Default value

windom.string.size = 13

Description

This variable sets the text size of string objects.

Parent group

windom.string

Programming guideline of WinDom

windom.string.font

Name

windom.string.font - WinDom button text font

Type

Variable

Syntax

windom.string.font = "Font name"

Default value

windom.string.font = "system font"

Description

This variable sets the text font of simple buttons.

Programming guideline of WinDom

windom.exit

Name

windom.exit - WinDom exit buttons configuration

Type

Group

Subgroups

windom.exit.color
windom.exit.font
windom.exit.size

Description

This group configures the look of WinDom exit button (default buttons in formulars).

Parent group

windom.exit

Programming guideline of WinDom

windom.exit.color

Name

windom.exit.color - WinDom exit button text color

Type

Variable

Syntax

windom.exit.color = AES colors index (0..15)

Default value

windom.exit.color = 1 (BLACK)

Description

This variable sets the text color of exit buttons.

Parent group

windom.exit

Programming guideline of WinDom

windom.exit.size

Name

windom.exit.size - WinDom exit button text size

Type

Variable

Syntax

windom.exit.size = Font size (in point)

Default value

windom.exit.size = 13

Description

This variable sets the text size of exit buttons.

Parent group

windom.exit

Programming guideline of WinDom

windom.exit.font

Name

windom.exit.font - WinDom exit button text font

Type

Variable

Syntax

windom.exit.font = "Font name"

Default value

windom.exit.font = "system font"

Description

This variable sets the text font of exit button.

Parent group

windom.exit

Programming guideline of WinDom

windom.menu

Name

windom.menu - WinDom menu configuration

Type

Group

Subgroups

windom.menu.color
windom.menu.font
windom.menu.size
windom.menu.effect
windom.menu.scroll

Description

This group configures the look of WinDom menu items/title objects and the window menu feels.

Parent group

windom.menu

Programming guideline of WinDom

windom.menu.color

Name

windom.menu.color - WinDom menu items text color

Type

Variable

Syntax

windom.menu.color = AES colors index (0..15)

Default value

windom.menu.color = 1 (BLACK)

Description

This variable sets the text color of menu items.

Parent group

windom.menu

Programming guideline of WinDom

windom.menu.size

Name

windom.menu.size - WinDom menu items text size

Type

Variable

Syntax

windom.menu.size = Font size (in point)

Default value

windom.menu.size = 13

Description

This variable sets the text size of menu itmes.

Parent group

windom.menu

Programming guideline of WinDom

windom.menu.font

Name

windom.menu.font - WinDom menu items text font

Type

Variable

Syntax

windom.menu.font = "Font name"

Default value

windom.menu.font = "system font"

Description

This variable sets the text font of menu items.

Parent group

windom.menu

Programming guideline of WinDom

windom.menu.effect

Name

windom.menu.effect - window menu selecting flashing effect.

Type

Variable

Syntax

windom.menu.effect = number of flashs or zero (no flashing effect)

Default value

windom.menu.effect = 3

Description

This variable sets the flashing effect when the user selects an item in a window menu.

Parent group

windom.effect

Programming guideline of WinDom

windom.menu.scroll

Name

windom.menu.scroll - add or remove the menu scroller widget.

Type

Variable

Syntax

windom.menu.scroll = {true|false}

Default value

windom.menu.scroll = false

Description

A true value makes appears the scroller widget in the window menu.

Parent group

windom.menu

Programming guideline of WinDom

windom.window

Name

windom.window - Window parameters.

Type

Group

Subgroups

windom.window.bg
windom.window.center
windom.window.effect

Description

All window related parameters are grouped here.

Parent group

windom

Programming guideline of WinDom

windom.window.bg

Name

windom.window.bg - Window background (workspace) parameters

Type

Group

Subgroups

windom.window.bg.color
windom.window.bg.pattern
windom.window.bg.style

Description

This group configures the look of windows background if the WindClear() function is used to draw the window background.

Parent group

windom.window

Programming guideline of WinDom

windom.window.bg.color

Name

windom.window.bg.color - set the color of window background.

Type

Variable

Syntax

windom.window.bg.color = VDI colors index

Default value

windom.window.bg.color = 0 (WHITE)

Description

This variable sets the color used to draw the window background. (the value is given to vsf_color()).

Related function

WindClear()

Parent group

windom.window.bg

Programming guideline of WinDom

windom.window.bg.pattern

Name

windom.window.bg.pattern - set the pattern type of window background.

Type

Variable

Syntax

windom.window.bg.pattern = VDI pattern index (0,1,2,3)

Default value

windom.window.bg.pattern = 1 (FIS_SOLID)

Description

This variable sets the type of pattern used to draw the window background. (the value is given to vsf_interior()). Different pattern type are :

- 0 (FIS_HOLLOW) : hollow interior
- 1 (FIS_SOLID) : solid interior
- 2 (FIS_PATTERN) : pattern fill
- 3 (FIS_HATCH) : hatched fill

The FIS_PATTERN and FIS_HATCH modes are controled by the variable windom.window.bg.style. The FIS_HOLLOW and FIS_SOLID modes don't depend on windom.window.bg.style.

Related function

WindClear()

Parent group

windom.window.bg

Programming guideline of WinDom

windom.window.bg.style

Name

windom.window.bg.style - set the VDI style of window background.

Type

Variable

Syntax

windom.window.bg.style = VDI style pattern index or VDI style hatched index.

Default value

windom.window.bg.style = 8

Description

This variable sets the style used to draw the window background. (the value is given to vsf_style()). The type of style depends on the value of variable windom.window.bg.pattern.

Related function

WindClear()

Parent group

windom.window.bg

Programming guideline of WinDom

windom.window.center

Name

windom.window.center - define how windows and formulars are centered.

Type

Variable

Syntax

windom.window.center = {screen|mouse|form|upleft|upright|dnleft|dnright}

Default value

windom.window.center = screen

Description

This variable defines how windows (and formulars) are centered or, in a general way, defines how the output of GrectCenter() is computed. If a center request is given to WindOpen() (i.e. x=-1, y=-1), WindOpen() computes the position with GrectCenter(). The results depends on the value of windom.windom.center. Possible values are:

- screen: the window is centered in the desktop,
- mouse: the window is centered around the mouse,
- form: the window is centered using the form_center() function, this function can be controlled by Let's Them Fly. Using this mode allows you to have forms and windows opened like non WinDom applications.
- upleft: the window is displayed in the up left corner of the desktop.
- dnleft: the window is displayed in the down left corner of the desktop.
- upright: the window is displayed in the up right corner of the desktop.
- dnright: the window is displayed in the down right corner of the desktop.

Related functions

GrectCenter(), WindOpen(), FormCreate(), FormBegin(), FormWindBegin()

Parent group

windom.window

Programming guideline of WinDom

windom.window.effect

Name

windom.window.effect - window graphic effects.

Type

Variable

Syntax

windom.window.effect = {true|false}

Default value

windom.window.effect = true

Description

This variable defines if a graphic effect is produces when a window is opened or closed. This variable is linked to the **WS_GRAFGROW** bit of the *flags* field of window descriptor.

Related functions

WindOpen(), WindClose().

Parent group

windom.window

Programming guideline of WinDom

windom.version

Name

windom.version - display the version number of WinDom

Type

Variable

Syntax

windom.version = {true|false}

Default value

windom.version = false

Description

If this variable is set to true, the application will display an alert box containing the version number of WinDom when the application is started (ApplInit()) or when EvntWindom() receives the **AP_LOADCONF** message.

Related function

ApplInit(), EvntWindom()

Parent group

windom

Programming guideline of WinDom

windom.popup

Name

windom.popup - Menu popup configuration

Type

Group

Subgroups

windom.popup.border
windom.popup.color
windom.popup.framec
windom.popup.pattern
windom.popup.relief
windom.popup.window

Description

This group configures the look and feel of menu popup.

Parent group

windom

Programming guideline of WinDom

windom.popup.border

Name

windom.popup.border - define the menu popup border size.

Type

Variable

Syntax

windom.popup.border = -4 ... 4

Default value

windom.popup.border = 2

Description

If a popup is displayed in **P_LIST** mode, this variable defines the menu popup border size. A negative value means that the border is exterior of the object menu.

Related function

MenuPopUp() with **P_LIST** mode.

Parent group

windom.popup

Programming guideline of WinDom

windom.popup.color

Name

windom.popup.color - define the menu popup background color.

Type

Variable

Syntax

windom.popup.color = AES color index (0..15)

Default value

windom.popup.color = 0 (WHITE)

Description

If a popup is displayed in **P_LIST** mode, this variable defines the menu popup background color.

Related function

MenuPopUp() with **P_LIST** mode.

Parent group

windom.popup

Programming guideline of WinDom

windom.popup.framec

Name

windom.popup.framec - define the color of menu popup frame

Type

Variable

Syntax

windom.popup.framec = AES color index (0..15)

Default value

windom.popup.framec = 1 (BLACK)

Description

If a popup is displayed in **P_LIST** mode, this variable defines the color of menu popup frame.

Related function

MenuPopUp() with **P_LIST** mode.

Parent group

windom.popup

Programming guideline of WinDom

windom.popup.pattern

Name

windom.popup.pattern - define the AES pattern of menu popup background

Type

Variable

Syntax

windom.popup.pattern = 0 .. 7

Default value

windom.popup.pattern = 0

Description

If a popup is displayed in **P_LIST** mode, this variable defines the pattern of menu popup background.

Related function

MenuPopUp() with **P_LIST** mode.

Parent group

windom.popup

Programming guideline of WinDom

windom.popup.relief

Name

windom.popup.relief

Type

Variable

Syntax

windom.popup.relief = {true|false}

Default value

windom.popup.relief = true

Description

If a popup is displayed in **P_LIST** mode, this variable defines if the popup is displayed with a relief effect.

Related function

MenuPopUp() with **P_LIST** mode.

Parent group

windom.popup

Programming guideline of WinDom

windom.popup.window

Name

windom.popup.window -

Type

Variable

Syntax

windom.popup.window = {true|false}

Default value

windom.popup.window = false

Description

This variable defines if a menu popup is displayed in a modal window and handled by a modal window formular (true value) or if a menu is handled by a classic formular (false value).

Related function

MenuPopUp()

Parent group

windom.popup

Programming guideline of WinDom

windom.fsel

Name

windom.fsel - File selector configuration

Type

Group

Subgroups

windom.fsel.path
windom.fsel.mask
windom.fsel.fslx

Description

This grup is devoted to the configuration of the file selector.

Parent group

windom

Programming guideline of WinDom

windom.fsel.path

Name

windom.fsel.path

Type

Variable

Syntax

windom.fsel.path = "path 1;path 2;..."

Default value

windom.fsel.path = NULL

Description

This variable defines a list of directories appearing in the file selector (if the system support it). Each directory has be delimited by a `;' character.

Related function

FselInput()

Parent group

windom.fsel

Programming guideline of WinDom

windom.fsel.mask

Name

windom.fsel.mask

Type

Variable

Syntax

windom.fsel.mask = "mask 1;mask 2;..."

Default value

windom.fsel.mask = NULL

Description

This variable defines a list of file mask appearing in the file selector (if the system support it). Each directory has be delimited by a ; character.

Related function

FselInput()

Parent group

windom.fsel

Programming guideline of WinDom

windom.fsel.fslx

Name

windom.fsel.fslx

Type

Variable

Syntax

windom.fsel.fslx = {true|false}

Default value

windom.fsel.fslx = true

Description

Related function

FselInput()

Parent group

windom.fsel

Programming guideline of WinDom

windom.iconify

Name

windom.iconify - configuration of iconified window.

Type

Group

Subgroups

windom.iconify.geometry

Description

This group is devoted to the configuration of icon windows. Currently, one variable is available.

Parent group

windom

Programming guideline of WinDom

windom.iconify.geometry

Name

windom.iconify.geometry - set the icon window size

Type

Variable

Syntax

windom.iconify.geometry = w,h (size in pixels)

Default value

windom.iconify.geometry = 72,72

Description

This variable sets the icon window size. It is working only if ICFS is present.

Related function

WindSet(WF_ICONIFY);

Parent group

windom.iconify

Programming guideline of WinDom

windom.mform

Name

windom.mform - configuration of modal formular.

Type

Group

Subgroups

windom.mform.widget

Description

This group is devoted to modal formular configuration.

Parent group

windom

Programming guideline of WinDom

windom.mform.widget

Name

windom.mform.widget - define the widgets of modal formular

Type

Variable

Syntax

windom.mform.widget = <hexadecimal value>

Default value

windom.mform.widget = 0x0009 (MOVER+NAME)

Description

This variable sets the window widget of modal formulars. It is an hexadecimale value, a bit field on the following values:

- NAME (0x1)
- CLOSER (0x2)
- FULLER (0x4)
- MOVER (0x8)
- INFO (0x10)
- SIZER (0x20)
- UPARROW (0x40)
- DNARROW (0x80)
- VSLIDE (0x100)
- LFARROW (0x200)
- RTARROW (0x400)
- HSLIDE (0x800)
- SMALLER (0x400)

Related function

WindFormBegin()

Parent group

windom.mform

Programming guideline of WinDom

windom.shortcut

Name

windom.shortcut - configuration of keyboard shortcuts

Type

Group

Subgroups

windom.shortcut.color

Description

This group configures the keyboard shortcut appearing in extended objets in formulars and toolbar.

Parent group

windom

Programming guideline of WinDom

windom.shortcut.color

Name

windom.shortcut.color - set the color of keyboard shortcuts.

Type

Variable

Syntax

windom.shortcut.color = AES color index

Default value

windom.shortcut.color = 1 (BLACK)

Description

This variable set the color of keyboard shortcut in formulars and toolbars. A keyboard shortcut appears as an underlined letter in a object label.

Parent group

windom.shortcul

Programming guideline of WinDom

windom.debug

Name

windom.debug - trace windom program

Type

Variable

Syntax

windom.debug = {debug|log|alert}[, path]

Default value

Not defined

Description

windom.debug defines the way debug() works. If the variable is not defined, debug() has no action. If the variable is set to 'alert', traces are displayed in an alert box. If the variable is set to 'log', the traces are written in a log file, a second parameter is required describing the path of the log file. If the variable is set to 'debug', traces are displayed using the WinDom DEBUG program (supplying in the WDK). A second parameter is required describing the path of the DEBUG program.

Related function

debug()

Parent group

windom

Programming guideline of WinDom

windom.relief

Name

windom.relief - relief effect.

Type

Group

Subgroups

windom.relief.color
windom.relief.mono

Description

This group configures the relief effect of extended object.

Parent group

windom

Programming guideline of WinDom

windom.relief.color

Name

windom.relief.color - set the relief color

Type

Variable

Syntax

windom.relief.color = AES color index

Default value

windom.relief.color = 8 (LIGHT GRAY)

Description

This variable sets the color used to draw the object with a relief effect when the screen supports 16 colors or more.

Parent group

windom.relief

Programming guideline of WinDom

windom.relief.mono

Name

windom.relief.mono -

Type

Variable

Syntax

windom.relief.mono = AES style index

Default value

windom.relief.mono = 0

Description

This variable sets the color used to draw the object with a relief effect when the screen is monochrome (actually for resolution with less of 16 color). As the resolution is monochrome, WinDom uses an AES motif style.

Parent group

windom.relief

Programming guideline of WinDom

windom.xlongbox

Name

windom.xlongbox - WinDom XLONGBOXTEXT object configuration

Type

Group

Subgroups

windom.xlongbox.color
windom.xlongbox.font
windom.xlongbox.size

Description

This group configures the look of XLONGBOXTEXT (extended objects).

Parent group

windom

Programming guideline of WinDom

windom.xtedinfo

Name

windom.xlongbox - WinDom XLONGBOXTEXT object configuration

Type

Group

Subgroups

windom.xlongbox.color
windom.xlongbox.font
windom.xlongbox.size

Description

This group configures the look of XLONGBOXTEXT (extended objects).

Parent group

windom

Programming guideline of WinDom

windom.

Name

Type

Syntax

Default value

Subgroups

Description

Related function

Programming guideline of WinDom

General index of variables

Programming guideline of WinDom

WinDom Programming User Reference

Programming guideline of WinDom

Application library

Programming guideline of WinDom

Application library

Programming guideline of WinDom

ApplInit()

NAME

ApplInit - AES and WinDom initialization.

PROTOTYPAGE

int ApplInit(**void**);

PARAMETERS

return: AES application handle (AES-id).

DESCRIPTION

This function replaces the appl_init() AES function. The AES and WinDom environements are iniatilized. The windom configuration file is read in order to setup the global windom variables.

SEE ALSO

ApplExit(), WinDom configuration.

Programming guideline of WinDom

ApplExit()

NAME

ApplExit - Terminate a WinDom session.

PROTOTYPAGE

int ApplExit(**void**);

PARAMETRES

return: error code.

DESCRIPTION

This function is the last call of a WinDom program. It replaces the appl_exit() function and release the memory reserved by the ApplInit() function.

SEE ALSO

ApplInit()

Programming guideline of WinDom

ApplName()

NAME

ApplName - returns the name of a GEM process.

PROTOTYPAGE

int ApplName(**char** *name, **int** id);

PARAMETERS

name:
name of the process (a 8-byte buffer),

id:
AES handle of the process,

return:
1 if process found, 0 else.

DESCRIPTION

This function gets the name of a GEM process using its AES process handle. This function uses the appl_search() function. If this function is not avalaible, ApplName() returns always 0. In this case the **AES4_APPSEARCH** bit of the *app.aes4* is 0.

SEE ALSO

appl_search(), appl_find().

Programming guideline of WinDom

ApplWrite()

NAME

ApplWrite - send a message to a GEM process.

PROTOTYPAGE

```
/* Prototype for 16 bits compilers */
int ApplWrite( int to, int msg, ...);
/* Prototype for 32 bits compilers */
int ApplWrite( int to, int msg, int w3, int w4, int w5, int w6, int w7);
```

PARAMETERS

to:
AES id of the targeted process,

msg:
message number to send,

...:
these parameters should fill the words 3 to 7 of the AES message pipe.

return:
the value returned by appl_write().

DESCRIPTION

This function is just an usefull call to the appl_write() AES function. It replaces the obsolet snd_msg() WinDom function. This function have two implementations : one for compilers which have a integer size of 16 bits and one for compilers wich have a interger size of 32 bits. For the second one, all pointer parameters should be absolutely encapsuled by the macro function ADR() because integer parameter and pointer parameter have the same 32 bits size. So we use the fixe prototype of 5 integer parameters to prevent during the compilation this king of error.

EXAMPLE

Instead of write :

```
{
    int pipe[8];
    char p[] = "C:\\NEWDESK.INF";

    pipe[0] = VA_START;
    pipe[1] = app.id;
    pipe[2] = 0;
    pipe[3] = strcpy( app.pipe, *(char**) & p);
    appl_write( id_target, 16, pipe);
}
```

just write:

```
ApplWrite( id_target, VA_START, ADR(strcpy(app.pipe,"C:\\NEWDESK.INF")), 0, 0);
```

Macro ADR() is required by 32-bits compilers but not for 16-bits. However, to increase the portability, we recommend to use ADR() macro function for pointer arguments. *app.pipe* is just a buffer in global memory reserved by WinDom and used for communications with extern GEM application. It is not required for internal communications.

SEE ALSO

appl_write(), snd_rdw(), Galloc().

Programming guideline of WinDom

ApplControl()

NAME

ApplControl - control of GEM process.

PROTOTYPAGE

int ApplControl(**int** ap_cid, **int** ap_cwhat);

PARAMETERS

ap_cid:
handle of the targeted application,

ap_cwhat:
mode :

- APC_HIDE: mask the application,
- APC_SHOW: show the application,
- APC_HIDENOT: no implemented yet,
- APC_TOP: no implemented yet.

return:
0 if error >0 else.

DESCRIPTION

MagiC and Naes have simulars but not identical functions to control the GEM process. This function tries to unify these calls.

BUGS

Only the **APC_HIDE** mode works correctly (used by EvntWindom()).

SEE ALSO

appl_control(), EvntWindom().

Programming guideline of WinDom

ApplSet()

NAME

ApplSet - Set application parameters.

PROTOTYPAGE

int ApplSet(**int** mode, ...);

PARAMETERS

mode:
see table below,

...
depend on mode value, see table below,

return:
0 if no error.

DESCRIPTION

ApplSet() sets global application parameters. When ApplInit() is called, the WinDom configuration is read. However, the developper can set its own settings using ApplSet(). The general call of ApplSet() is :

```
int par1, par2, par3, par4;
ApplSet( mode, par1, par2, par3, par4);
```

Usage of *par1*, *par2*, *par3* and *par4* depends on *mode* value. The following table lists the different mode of ApplSet(). Each mode matches one or more variables inside the WinDom Configuration file. Correspondance with these variables are printer under the mode name. The **DEFVAL** value does not change the value. For example :

```
/* Set only the color of string objects */
ApplSet( APS_STRSTYLE, DEFVAL, DEFVAL, RED);
```

ApplSet() mode

Mode/Variable	Description	Parameters
APS_ICONSIZE	Set the window icon	par1 = icon width
windom.iconify	size.	par2 = icon height
APS_FLAG	Set the application	par1 = flags. Flags are:
	flags.	
		par1 = Bit to set/unset
		par2 = TRUE/FALSE
		Possible bits are :
windom.evnt.button		FLG_KEYMOUSE keybord event
		on mouse
windom.evnt.keybd		FLG_BUTMOUSE mouse event
		on mouse
no variable		FLG_NOPAL disable color
		palette handling
windom.menu.scroll		FLG_MNSCRL enable menu
		scroller widget
no variable		FLG_NOKEYMENU disable menu
		shortcuts handling
windom.fsel.fslx		FLG_NOMGXFSEL disable MagiC
		file selector.
APS_WINBG	Set the window	par1 = VDI color index
windom.window.bg	background style.	par2 = VDI type of pattern
		par3 = VDI style index
APS_KEYCOLOR	Set the color of	par1 = VDI color index
windom.shortcut	keyboards shortcut.	
APS_STRSTYLE	Set the style of	par1 = GDOS font id
windom.string	string object.	par2 = font size
		par3 = VDI color index
APS_BUTSTYLE	Set the style of	as APS_STRSTYLE
windom.button	BUTTON object.	
APS_EXITSTYLE	Set the style of	as APS_STRSTYLE
windom.exit	EXIT object.	
APS_TITLESTYLE	Set the style of	as APS_STRSTYLE
windom.menu	TITLE object.	
APS_3DEFFECT	Control object 3D	par1 = windom.relief.color
windom.relief	effect.	par2 = windom.relief.mono
APS_MENUEFFECT	Control the flashing	par1 = windom.menu.effect
windom.menu	effect of window	

	menu selection.	
APS_BUBBLESTYLE	Set the style of	par1 = GDOS font id
windom.bubble	bubble help.	par2 = font size
APS_POPUPSTYLE	Set the style of	par1 = background AES color index
windom.popup	menu popup.	par2 = border AES color index
		par3 = frame AES color index
		par4 = background AES style index
APS_POPUPWIND	Use menu popup window	par1 = TRUE/FALSE
windom.popup		
APS_WINDOWS	General window	par1 = window/dialog centering
windom.window	parameters.	Possible values are :
		CENTER center in screen
		WMOUSE center on mouse position
		UP_RIGHT upper right screen corner
		UP_LEFT upper left screen corner
		DN_RIGHT down right screen corner
		DN_LEFT down left screen corner
		FCENTER center with form_center()
		par2 = grafic effect when open and
		close window (TRUE/FALSE)
		par3 = window wigdet of modal
		dialog (as in WindCreate()).

SEE ALSO

ApplGet(), WinDom Configuration.

Programming guideline of WinDom

ApplGet()

NAME

ApplGet - Get application parameters.

PROTOTYPAGE

int ApplGet(**int** mode, ...);

PARAMETERS

mode:
see table below,

...
depend on mode value,

return:
0 if no error.

DESCRIPTION

ApplGet() returns global application parameters. ApplGet() performed the inverse action of ApplSet(). The general call of ApplGet() is :

```
int par1, par2, par3, par4;
ApplSet( mode, &par1, &par2, &par3, &par4);
```

Usage of *par1*, *par2*, *par3* and *par4* depends on *mode* value. For details about this mode, read manual of ApplSet(). The **NULL** value can be used if a parameter hasnot to be read.

```
/* Get only the color of string objects */
int color;
ApplSet( APS_STRSTYLE, NULL, NULL, &color);
```

See ApplSet() table which list all avalaible modes.

SEE ALSO

ApplSet(), WinDom Configuration.

Programming guideline of WinDom

AV library

Programming guideline of WinDom

AvInit()

NAME

AvInit - Initialization of the AV protocol.

PROTOTYPAGE

int AvInit(**char** *name, **int** status, **long** idle)

PARAMETERS

name:
name of client (with an appl_find() format),

status:
actions supported by client : a bit field of values listed in the AV client states table,

idle:
time idle of the server,

return:
AES id of the AV server or error code:

-1
server not found,

-2
server doesn't not supporte AV protocol.

DESCRIPTION

This function initialize the AV session between the client and the AV server and waits for the answer of the server. Use the AvStatus() function to know the actions supported by the server. The AV server is identifiate by reading the environ variable AVSERVER. If this variable is not defined, AvInit() tries the following process: 'AVSERVER', 'THING', 'GEMINI' then the desktop application.

AvInit() declares to the server the actions supported by the client (our application). The AV client states table gives the diferents values possible. Among these values, the **A_QUOTE** value is very importante. It means that the client supported the quoted filename : when a filename containt a space character, the complete string is surronded by a quote character. For example, the string "The World" is quoted: "'The World'". Use the AvStrfmt() function to quoted or unquoted the strings.

VARIABLES

The *app.avid* containts the AV server AES id.

SEE ALSO]
AvExit(), AvStrfmt(), AvStatus(), AvServer().

Programming guideline of WinDom

AvExit()

NAME

AvExit - close an AV session opened with AvInit().

PROTOTYPAGE

void AvExit(**void**);

DESCRIPTION

Before terminate a client, the AV session must be absolutely closed with this function.

VOIR AUSSI

AvInit()

Programming guideline of WinDom

AvServer()

NAME

AvServer - returns the AV server states.

PROTOTYPAGE

char *AvServer(**void**);

PARAMETERS

return:
name of the AV server.

DESCRIPTION

This function returns the name of the AV server if the AV session was succefully opened with the AvInit() function. The AV server GEM identifier is given by the global variable *app*.*avid* .

SEE ALSO

AvInit(), AvStatus().

Programming guideline of WinDom

AvStatus()

NAME

AvStatus - returns the AV server states.

PROTOTYPAGE

int *AvStatus(**void**);

PARAMETERS

return:
pointer to a 3-integer array.

DESCRIPTION

This function returns the actions supported by the AV server, if the AV session was succefully opened with the AvInit() function. AvStatus returns a pointer to a 3-integer array. Each value of this array is a bit field whose the values are listed in the AV server states table.

SEE ALSO

AvServer(), AvInit().

Programming guideline of WinDom

AvWaitfor()

NAME

AvWaitfor - Wait for a message.

PROTOTYPAGE

int AvWaitfor(**int** msg, **INT16** *buf, **long** idle);

PARAMETERS

msg:
message to wait for,

buf:
8-word buffer,

idle:
time idle,

return:
1 if the message is received, 0 else.

DESCRIPTION

AvWaitfor() waits for a specific message (**MU_MESAG** event). Typically, it is the answer of an AV request sent to the AV server. If other messages occur, these messages are not lost but are resent to the application.

DRAWBACK

AvWaitfor() does not make use of EvntWindom(), just evnt_multi().

EXAMPLE

```
/* send an AV request and wait the answer */

int exec_prog( char *prg, char *cmd) {
        ApplWrite( app.avid, AV_STARTPROG, prg, cmd, 0);
        return AvWaitfor( VA_PROGSTARTED, evnt.buf, 1000);
}
```

Programming guideline of WinDom

AvStrfmt()

NAME

AvStrfmt - filename format for AV usage.

PROTOTYPAGE

char *AvStrfmt(**int** mode, **char** *src);

PARAMETERS

src:
filename,

mode:

1:
if needed, the filename will be unquoted,

0:
if needed, the filename will be quoted.

return:
the quoted/unquoted filename (dynamically created).

DESCRIPTION

With some modern filesystems, the filename can contain space characters. In this case, the AV protocol ask to `quote' a filename, i.e. quote characters are added at the beginning and the end of the filename. So, it is very important to use this function all the time when you send or received AV request with filename parameters.

Please note, some desktops use the quoting filename with drag'n drop and argv protocol.

Programming guideline of WinDom

BubbleGEM library

Programming guideline of WinDom

BubbleCall()

NAME

BubbleCall - Display a bubble help.

PROTOTYPAGE

int BubbleCall(**char** *help, **int** x, **int** y);

PARAMETERS

help:
pointeur to the string to display in a bubble,

x,y:
coordinates of the bubble (use the mouse position),

return:
0 si no error,

ERROR CODE

BubbleCall() returns a code error.

0:
no error,

-1:
BubbleGEM not in memory,

-2:
the environ variable 'BUBBLE=' or 'BUBBLEGEM=' are incorrects,

-3:
no more memory,

DESCRIPTION

BubbleCall() sends a message to BUBBLE.APP to display a bubble help. If BUBBLE.APP is not loaded, BubbleCall() tries to load it using the PATH or BUBBLE or BUBBLEGEM environ variables. This function just display a bubble help. It is possible to attach a bubble help to an object from a formular or a toolbar and display them systematically. For that purpose, see the BubbleAttach() and BubbleEvnt() functions.

REMARKS

A '|' character forces a carriage return inside the bubble help.

The AES should not be stopped, that is the case with classic formulars which make use of wind_update() function, when the BubbleCall() function is invoked. However, it is possible to call BubbleGEM from a classic formular with the BubbleModal() function.

SEE ALSO

(!url [The BubbleGEM documentation] [BUBBLE.HYP]), BubbleAttach(), BubbleEvnt(), BubbleFree(), BubbleConf().

Programming guideline of WinDom

BubbleAttach()

NAME

BubbleAttach - Link a bubble help to an object.

PROTOTYPAGE

int BubbleAttach(OBJECT *tree, **int** index, **char** *help);

PARAMETERS

tree:
object tree address,

index:
object index,

help:
address of string to display in the bubble (the string is not duplicated),

return:
0 if no error, -1 if memory error.

DESCRIPTION

This function links a bubble help to an objet in a window formular or a toolbar. The bubble will be displayed by the BubbleEvnt() function, typically after a timer event or a right mouse button event. When the program terminates, the bubbles should be free up with the BubbleFree() function.

SEE ALSO

BubbleEvnt(), BubbleFree(), BubbleFind().

Programming guideline of WinDom

BubbleEvnt()

NAME

BubbleEvnt - display the bubble help defined by BubbleAttach().

PROTOTYPAGE

int BubbleEvnt(void);

PARAMETERS

valeur de retour:

- 0 if bubble is not found,
- WS_FORM if bubble is found in a dialog box,
- MW_TOOLBAR if bubble is found in a toolbar.

DESCRIPTION

This function find the object pointed by the mouse sprite. If the an object is found and if a bubble help is linked to this object (with BubbleAttach()) the bubble help is displayed. BubbleEvnt() works only with window formular or toolbar.

EXAMPLE

```
/*
 * Example of BubbleGEM support
 * with WinDom
 */
#include <windom.h>

void RightButton( void) {
   if( evnt.mbut & 0x2)
            BubbleEvnt();
}

int main(void) {
   OBJECT *tree;

   ApplInit();
   RsrcLoad( "TEST.RSC");
   rsrc_gaddr( 0, FORM1, &tree);

   /* Link the bubble help to objects ... */
   BubbleAttach( tree,  0, "Formular background"); /* FORM1 */
   BubbleAttach( tree, OK, "An exit button");      /* FORM1 */

   /* Create form */
   FormCreate( tree, MOVER|NAME, NULL, "test", NULL, 1, 0);
```

```
        /* Handle globally the MU_BUTTON event */
        EvntAttach( NULL, WM_XBUTTON, RightButton);

        /* On gère les clicks souris GAUCHE et DROIT */
        evnt.bclick = 258;
        evnt.bmask = 0x1|0x2;
        evnt.bstate = 0;

        do
                EvntWindom( MU_MESAG|MU_BUTTON);
        while( wglb.first);

        BubbleFree();  /* release the bubbles */
        RsrcFree();
        ApplExit();
        return 0;
}
```

SEE ALSO

BubbleAttach(), BubbleFree(), BubbleFind(), W_FORM.

Programming guideline of WinDom

BubbleFree()

NAME

BubbleFree - release memory reverved by BubbleAttach().

PROTOTYPAGE

void BubbleFree(**void**);

SEE ALSO

BubbleAttach().

Programming guideline of WinDom

BubbleFind()

NAME

BubbleFind - Find a bubble linked to an object.

PROTOTYPAGE

int BubbleFind(OBJECT *tree, **int** index, **char** **help)

PARAMETERS

tree:
address of object tree,

index:
object index,

help:
address of string linked,

return:
1 if a bubble is found, 0 else.

DESCRIPTION

BubbleFind() is called by BubbleEvnt() to find the text to display in a bubble help. It can be used to find a bubble and display it with BubbleCall() if you are not interesting to use BubbleEvnt().

SEE ALSO

BubbleCall(), BubbleEvnt(), BubbleAttach().

Programming guideline of WinDom

BubbleConf()

NAME

BubbleConf - Local configuration of BubbleGEM.

PROTOTYPAGE

void BubbleConf(**int** delay, **int** flag)

PARAMETERS

flags:
bit field:

BGC_FONTCHANGED
(0x01) :

BGC_NOWINSTYLE
(0x02) :

BGC_SENDKEY
(0x04) :

BGC_DEMONACTIVE
(0x08) :

BGC_TOPONLY
(0x10) :

DESCRIPTION

This function configures locally (i.e. only for the application) the BubbleGEM setting. For a global setting, the user will use the CPX dedicated to BubbleGEM parametrization. This function uses the 'BHLP' cookie to parametrize BubbleGEM.

SEE ALSO

ConfRead(), BubbleCall().

Programming guideline of WinDom

BubbleModal()

NAME

BubbleModal - Display a bubble help in a classic formular.

PROTOTYPAGE

void BubbleModal(**char** *help, **int** x, **int** y)

PARAMETERS

see BubbleCall() parameters.

DESCRIPTION

BubbleModal() is the equivalent function of BubbleCall() function in the case of classic formulars (i.e. a formular stopping AES events). This function works only from BubbleGEM R05. With former version of BubbleGEM, it is not possible to call BubbleGEM from a classic formular.

If BUBBLE.APP is not present in memory, BubbleModal() is not able to load it (because AES is stopped). So the BubbleGEM daemon can be launched prevously with the BubbletGet() function.

Because the form_do() function can not handle the right mouse button event, WinDom offers an alternative function to display systematically bubbles with BubbleAttach() and BubbleEvnt() functions : it is the BubbleDo() function. This function is not universal, for custom usage, write your own BubbleDo() function (looking at the BubbleDo() source code located in SRC\BUBBLE.C file of the WinDom Developer Kit (WDK) package. This function is simple, it uses the functions of the AES form library.

A complet example is given is the folder EXAMPLES\BUBBLE of the WDK package.

SEE ALSO

BubbleGet(), BubbleDo(), BubbleConf(), BubbleAttach().

Programming guideline of WinDom

BubbleGet()

NAME

BubbleGet - load the BubbleGEM daemon in memory.

PROTOTYPAGE

int BubbleGet(**void**)

PARAMETERS

retunr:
GEM id of BUBBLE.APP daemon

DESCRIPTION

BubbleGet() loads if needed BUBBLE.APP, the BubbleGEM daemon, in memory and returns its AES app-id. Its uses the AES environ variable 'BUBBLE=' and 'BUBBLEGEM=' to locate the program. A search in 'PATH=' folders is performed too. BubbleGet() is called because a BubbleModal() call.

SEE ALSO

BubbleModal(), BubbleCall().

Programming guideline of WinDom

BubbleDo()

NAME

BubbleDo - alternative FormDo() function for BubbleGEM.

PROTOTYPAGE

int BubbleDo(OBJECT *tree, **int** index)

PARAMETERS

see FormDo().

DESCRIPTION

see BubbleModal() documentation.

SEE ALSO

BubbleModal().

Programming guideline of WinDom

Configuration library

Programming guideline of WinDom

ConfRead()

NAME

ConfRead - read the configuration file and set the WinDom parameters.

PROTOTYPAGE

int ConfRead(**void**);

PARAMETERS

ConfRead() returns an error code:

0:
no error,

-1:
application not find in the configuration file,

-33:
configuration file not found.

DESCRIPTION

ConfRead() reads in the WinDom configuration file the settings of the application. Only WinDom parameters are set. This file can contain other parameters readable with the functions ConfGetLine() and ConfInquire(). These parameters are specific to the application.

This function is used by ApplInit() to initialize the WinDom settings and by EvntWindom() when it receives the **AP_CONF** message.

The WinDom configuration file is an unique text file grouping the configuration of all WinDom application in a similar way then the '.Xdefaults' X11 file for example. This file is typically located in the HOME folder. Actually, WinDom searchs this file in the following directories :

- the application directory,
- the $HOME\Defaults directory,
- the $HOME directory,
- the $ETCDIR directory,
- the $PATH directories (can be multiple),
- the 'C:\' directory

This file is nammed 'windom.cnf' or '.windomrc' if the file system supports the long name file.

BUGS

Each line has the following syntax :

keyword = value list ...

A space character is required between the keyword and the '=' character.

SEE ALSO

ApplInit(), EvntWindom(), ConfInquire(), ConfGetLine(), WinDom configuration.

Programming guideline of WinDom

ConfInquire()

NAME

ConfInquire - read an user variable in the configuration file.

PROTOTYPAGE

int ConfInquire(**char** *keyword, **char** *format, ...);

PARAMETERS

keyword:
name of variable to read,

format:
format of value (see below),

...:
address of variables to fill up,

return:
an error code :

-33:
configuration file not found,

-1:
variable not found,

>=0:
number of values read,

DESCRIPTION

ConfInquire() read a variable from the configuration file in the application area is defined or in the Default Settings area if defined. If the variable is not found or if no configuration area addressing the application is not found, the function returns -1.

The syntax of variable definition in the configuration file have the following structure:

```
keyword = value[, value[, ...]]
```

The '[]' notation means an optional argument. The *format* parameter have a similar syntax than printf(). Possible variable are :

%d:
16-bit integer,

%f:

single real (32-bit),

%c:
a character delimited by a quote ('),

%b:
a boolean variable (true, on, 1, false, off, 0),

%B:
the boolean value in set in a specific bit of the variable (see EXAMPLES),

%s:
a string. The string can be delimited by a double quote charactere (") if the string contains space characters.

%S:
equivalent to %s, it is an obsolet mode but kept for higher compatibility.

EXAMPLES In the WINDOM.CNF file:

```
appli.font.name = "Helvetica Bold"
appli.system.path = C:\APPLI\system\
appli.window.size = 400,300
appli.window.sizer = 'S'
appli.parameters.save = TRUE
appli.parameters.bubble = TRUE
```

In the application:

```
    void InitAppl( void)
    {
        char FontName[33], path[128];
        int width, height;
        char car;
#          define PARAM_SAVE       0x1
#          define PARAM_BUBBLE 0x2
        int param;

        if( ConfInquire( "appli.font.name", "%s", FontName) != 1)
            strcpy( FontName, "Times");
        if( ConfInquire( "appli.system.path", "%s", path) != 1)
            strcpy( path, "");
        if( ConfInquire( "appli.window.size", "%d,%d", &width, &height) != 2)
            width = height = 200;
        if( ConfInquire( "appli.window.sizer", "%s", &car) != 1)
            car = 'S';
        ConfInquire( "appli.parameters.save", "%B", &param,   PARAM_SAVE);
        ConfInquire( "appli.parameters.bubble", "%B", &param, PARAM_BUBBLE);
    }
```

BUGS

See ConfRead().

SEE ALSO

ConfRead(), ConfWrite(), ConfGetLine(), Windom configuration.

Programming guideline of WinDom

ConfGetLine()

NAME

ConfGetLine - read acces by line in the configuration file.

PROTOTYPAGE

int ConfGetLine(**char** *line);

PARAMETERS

line:
pointer to a buffer to store the line,

return:

0:
no more line to read,

positive value:
number of the line read.

DESCRIPTION

ConfGetLine() provides a line acces to read the configuration file. ConfInquire() can only read a line with the following format: variable = value. If the *string* parameter is NULL, the line pointer inside the file is set to the begin of the application area.

BUGS

string should pointer to a sufficially big buffer to store the line.

SEE ALSO

ConfInquire(), ConfRead(), ConfWrite(), WinDom configuration.

Programming guideline of WinDom

Cookies Library

Programming guideline of WinDom

get_cookie()

NAME

get_cookie() - Look for a cookie in the system cookiejar.

PROTOTYPAGE

int get_cookie(**long** cookie, **long** *value);

PARAMETERS

cookie:
cookie identifier to find,

value:
address of the cookie value. A **NULL** value is possible.

return:
1 if the cookie is found, 0 else.

DESCRIPTION

get_cookie() finds a cookie in the cookiejar. If the system does not support the cookie jar, the function returns always 0. It is possible to test the cookiejar availability with get_cookiejar(). The function returns the address of the cookie value. It is possible to use the NULL value if the cookie value is not used.

EXAMPLES

```
/* LDG in memory ? */
int is_ldg( void) {
   return get_cookie( 'LDGM', NULL);
}

/* MiNT version */
int mint_version( void) {
   long val;
   if( get_cookie( 'MiNT', &val))
           return val;
   else
           return 0;
}
```

SEE ALSO

get_cookiejar().

Programming guideline of WinDom

get_cookiejar()

NAME

get_cookiejar() - return the cookiejar address

PROTOTYPAGE

long *get_cookiejar(**void**);

PARAMETERS

return: address of the system cookiejar or **NULL** value.

DESCRIPTION

get_cookiejar() returns address of the system cookie jar of a **NULL** value if the system does not support cookie jar.

Programming guideline of WinDom

new_cookie()

NAME

new_cookie() - insert a cookie in the cookiejar.

PROTOTYPAGE

int *new_cookie(**long** cookie, **long** value);

PARAMETERS

cookie:
cookie identifier to insert,

value:
cookie value to insert,

return:
0 if echec (cookiejar full), 1 if no error.

DESCRIPTION

new_cookie inserts a cookie in the cookiejar. If the cookiejar is full, new_cokie() is not able to reallocate the cookiejar. In this case you should resize the cookiejar with a patch such as INSJAR.

SEE ALSO

set_cookie().

Programming guideline of WinDom

set_cookie()

NAME

set_cookie() - set a cookie in the cookiejar.

PROTOTYPAGE

int *set_cookie(**long** cookie, **long** value);

PARAMETERS

cookie:
cookie identificator,

value:
new cookie value,

return:
0 si error (cookie not found), 1 if no error

DESCRIPTION

set_cookie() sets a new value of a cookie. If the cookie is not found, it is not created.

SEE ALSO

new_cookie().

Programming guideline of WinDom

Data library

Data library is devoted to handle windows' user data.

Programming guideline of WinDom

DataAttach()

NAME

DataAttach - Attach a data to a window.

PROTOTYPAGE

int DataAttach(WINDOW *win, **long** magic, **void** *data);

PARAMETERS

win:
targetted window,

magic:
magic number,

data:
address of data to attach,

return:
0 si no error or a negativ error code

DESCRIPTION

DataAttach() attaches an user data to a window. There are no limit (except the memory) to the numberof data. Data are stored in a list whose the window keeps the root item. Each item in the list, so each data, is identified by a magic number (as cookies in the system cookiejar). Some magic number are reserved by WinDom (because some predefined windows, such as form, use data). See WinDom header file (they have a WD_ prefix).

SEE ALSO

DataSearch(), DataDelete().

Programming guideline of WinDom

DataSearch()

NAME

DataSearch - Search a data attached to a window.

PROTOTYPAGE

void *DataSearch(WINDOW *win, **long** magic);

PARAMETERS

win:
targetted window,

magic:
magic number,

return:
data address or NULL if not found.

DESCRIPTION

The function is used to get a data attached to a window. For example the call :

```
W_FORM* form = DataSearch( win, WD_WFRM);
```

returns the formular data of a window.

SEE ALSO

DataAttach(), DataDelete().

Programming guideline of WinDom

DataDelete()

NAME

DataDelete - Delete a data attached to a window.

PROTOTYPAGE

int DataDelete(WINDOW *win, **long** magic);

PARAMETERS

win:
targetted window,

magic:
magic number,

return:
0 if no error or a negativ error code.

DESCRIPTION

The function is used to remove a data of a window. DataDelete() is typically call by the window destructor function.

EXAMPLE

```
/* Typical destroy function */
void WinDestroy( WINDOW *win) {
        MY_DATA *data;          /* associated magic number: 'MDTA' */

        /* Get data */
        data = DataSearch( win, 'MDTA');
        /* MY_DATA specific function to release data */
        free_mydata( data);
        /* Remove data */
        DataDelete( win, 'MDTA');
        ...
}
```

SEE ALSO

DataAttach(), DataSearch().

Programming guideline of WinDom

Event library

Programming guideline of WinDom

EvntWindom()

NAME

EvntWindom - GEM events handling.

PROTOTYPAGE

int EvntWindom(**int** event);

PARAMETRES

event:
bit field of event to handle,

return:
bit field of occured events.

DESCRIPTION

This function is the heart of WinDom. It replaces the AES evnt_multi() function (the EvntMulti() is already used by the Pure C AES bindings).

EvntWindom() is a little bit complex:

- it calls the evnt_multi() function,
- it handles the color palette depending of the topped window or the desktop palette if no topped window,
- it calls the good event function depending the event occured (**MU_MESAG**, **MU_BUTTON**, ...),
- window menu window, toolbar, keyboard shortcuts, specific WinDom features are handled. If needed, the AES special features (iconfications, bottom windows, untoppable and modal windows) are emulated,
- some new messages are eventually sent.

EvntWindom() can be parametrized (as evnt_multi()). The parametrization is performed in the gobal variable evnt. Some events return some additional informations. These informations are stored in the evnt variable too. This variable is a C-struct which have the following compostion:

```
typedef struct {
  /* parametrization variables
   ***************************/
    /* MU_TIMER parameters - see evnt_timer() */
    int lo_timer, hi_timer;
```

```
            /* MU_BUTTON parameters - see evnt_button() */
            int bclick, bmask, bstate;
            /* MU_M1  parameters */
            int m1_flag, m1_x, m1_y, m1_w, m1_h;
            /* MU_M2  parameters */
            int m2_flag, m2_x, m2_y, m2_w, m2_h;
          /* result variables
           ********************/
            /* MU_MESAG result */
            int buff[8];
            /* MU_BUTTON result - see evnt_button()*/
            int mx, my, mbut, mkstate;
            /* MU_KEYBD result - see evnt_keybd() */
            int keybd, nb_click;
} EVNTvar;
```

SEE ALSO

Event messages used by WinDom, EvntAttach(), evnt_multi().

Programming guideline of WinDom

EvntAttach()

NAME

EvntAttach - bind a function to a GEM event.

PROTOTYPAGE

int EvntAttach(WINDOW *win, **int** ev, **void** *proc);

PARAMETERS

win:
window targeted or NULL,

ev:
event to bind (see event list),

proc:
function address to bind.

DESCRIPTION

This function links a function to a GEM event. A GEM event is a button event, keyboard event, ... but a message event too. An event can be applied to a window or to the application and sometime both. Possible events are:

WM_XTIMER:
timer event (MU_TIMER),

WM_XBUTTON:
button event (MU_BUTTON),

WM_XKEYBD:
keyboard event (MU_KEYBD),

WM_XM1:
MU_M1 event,

WM_XM2:
MU_M2 event,

others:
others values address the GEM messages (WM_REDRAW, etc)

An event can be attached to a window or more, to the application or both window and application. If an event is previously defined, a new call of EvntAttach() on this event removes the old event link.

A function binded to an event has the following prototype :

```
void function( WINDOW *win);
```

where *win*is the descriptor of the targeted window or **NULL**if the event addresses the application.

EXAMPLES

Define a global button event:

```
EvntAttach( NULL, WM_XBUTTON, AppButton);
```

Define the button event of a window:

```
EvntAttach( win, WM_XBUTTON, WinButton);
```

Define the window destroy event:

```
EvntAttach( win, WM_DESTROY, WinDestroy);
```

Define a global event message (the application shutdown):

```
EvntAttach( NULL, AP_TERM, ApTerm);
```

SEE ALSO

EvntAdd(), EvntDataAttach(), EvntDelete(), EvntExec(), EvntFind(), EvntWindom().

Programming guideline of WinDom

EvntAdd()

NAME

EvntAdd - add a function in a GEM event bind.

PROTOTYPAGE

int EvntAdd(WINDOW *win, **int** ev, **void** *proc, **int** mode);

PARAMETERS

win:
window targeted or NULL,

ev:
event to bind (see event list),

proc:
function address to add,

mode:

EV_TOP:
add the function in top position,

EV_BOT:
add the function in bottom position,

DESCRIPTION

This function allows you to bind sereval different functions to a same event. Note that EvntAttach() can only bind one function to an event. Functions can be inserted in top position - mode = **EV_TOP** - (it will call in first) or in bottom position - mode = **EV_BOT** - (it will call in last). In general way, a function is added in bottom position. The **WM_DESTROY** event is often an exeception. As the window should be destroy in last, additionnal function making reference to the window should be call in top position. For usage of other parameters see the EventAttach() manual.

EXAMPLES

Windows have a default **WM_REDRAW** function (WindClear()). So prefer EventAdd() to EventAttach():

```
EvntAdd( win, WM_REDRAW, WinRedraw, EV_BOT);
```

Then, WindClear() will be firstly called then WinRedraw() will be called.

Windows have a default **WM_DESTROY** function (see WindCreate()) wich close, destroy the window and send an AP_TERM message if no more windows are in memory. A typical bind to **WM_DESTROY** is :

```
 EvntAdd( NULL, WM_DETROY, WinDestroy, EV_TOP);
/* and the Destroy function : */
void WinDestroy( WINDOW *win) {
        /* Free up data attached to window
        but not destroy the window */
}
```

SEE ALSO

EvntAttach(), EvntDelete(), EvntExec(), EvntFind(), EvntWindom().

Programming guideline of WinDom

EvntDataAttach()

NAME

EvntDataAttach - bind a function with data to a GEM event.

PROTOTYPAGE

int EvntDataAttach(WINDOW *win, **int** ev, **void** *proc, **void** *data);

PARAMETERS

win:
window targeted or NULL,

ev:
event to bind (see event list),

proc:
function address to bind,

data:
user data pointer.

DESCRIPTION

This function is similar to EvntAttach(). The difference is EvntDataAttach() binds a local pointer data to the object. This data is read by the binded function as a second parameter. The binded has the following prototype :

```
void function( WINDOW *win, void *data);
```

See EvntAttach() for a detailled description.

EXAMPLES

```
EvntDataAttach( NULL, WM_XBUTTON, AppButton, "Button event");
```

SEE ALSO

EvntDataAdd(), EvntAttach(), EvntDelete(), EvntExec(), EvntFind(), EvntWindom().

Programming guideline of WinDom

EvntDataAdd()

NAME

EvntDataAdd - add a function with data in a GEM event bind.

PROTOTYPAGE

int EvntDataAdd(WINDOW *win, **int** ev, **void** *proc, **void** *data, **int** mode);

PARAMETERS

win:
window targeted or NULL,

ev:
event to bind (see event list),

proc:
function address to add,

mode:

EV_TOP:
add the function in top position,

EV_BOT:
add the function in bottom position,

data:
user data pointer.

DESCRIPTION

This function is similar to EvntAdd() except it allows to bind an user data pointer with the function. The data is read by the binded function as a second parameter (see EvntDataAttach() manual). For detailled description, see EvntAddr().

SEE ALSO

EvntAttach(), EvntAdd(), EvntDataAttach(), EvntDelete(), EvntExec(), EvntFind(), EvntWindom().

Programming guideline of WinDom

EvntDelete()

NAME

EvntDelete - Delete an event.

PROTOTYPAGE

void EvntDelete(WINDOW *win, **int** ev);

PARAMETERS

win:
targeted window or NULL,

ev:
event to delete.

DESCRIPTION

EvntDelete() removes all functions binded to an event.

SEE ALSO

EvntAttach()

Programming guideline of WinDom

EvntClear()

NAME

EvntClear - removes all defined events.

PROTOTYPAGE

void EvntClear(WINDOW *win);

PARAMETERS

win: targeted window or NULL.

DESCRIPTION

EvntClear() removes all events defined to a window (if *win* parameter is non null) or to the application (if null).

SEE ALSO

EvntAttach()

Programming guideline of WinDom

EvntFind()

NAME

EvntFind - Find the first function binded to an event.

PROTOTYPAGE

void* EvntFind(WINDOW *win, **int** ev);

PARAMETERS

win:
targeted window or NULL,

ev:
event to find,

return:
function address.

DESCRIPTION

EvntFind() returns the first function address binded to a message for a window or for the application. If the event is unbinded, EvntFind() returns NULL. Currently, it is not possible to have the list of function binded to a message. This function is just used to know if a message is defined.

SEE ALSO

EvntAttach(), EvntAdd(), EvntExec().

Programming guideline of WinDom

EvntExec()

NAME

EvntExec - Execute all functions binded to an event.

PROTOTYPAGE

int EvntExec(WINDOW *win, **int** ev);

PARAMETRES

win:
targeted window or NULL,

ev:
event to exec,

return:
TRUE if the event function is found and correctly executed.

DESCRIPTION

EvntExec() finds all event functions binded to the message *ev* and executes them if found. Note **WM_REDRAW** cannot be executed by EvntExec(), we have to used EvntRedraw for that.

There is an important difference between ApplWrite() and EvntExec() even if the result seen the same. When you send a message with ApplWrite(), we give the control to AES before to execute the message. With EvntExec(), events are executed directly. Some actions, as closing a window for example, can confuses AES if they are executed directly. Each time is possible, prefer ApplWrite() rather than EvntExec(). Give the control to AES increases the multasking performance.

SEE ALSO

EvntFind(), EvntAttach(), EvntDisable(), EvntEnable(), EvntRedraw().

Programming guideline of WinDom

EvntRemove()

NAME

EvntRemove - Remove one function binded to an event.

PROTOTYPAGE

int EvntRemove(WINDOW *win, **int** ev, **void** *proc);

PARAMETRES

win:
targeted window or NULL,

ev:
event to find,

proc:
function to remove.

return:
TRUE if the function is found and correctly removed.

DESCRIPTION

EvntRemove() removes one function binded to an event. It is different to EventDelete() which removes all functions binded to an same event. Because a same function can be binded to different event, you should give an event (*ev* parameter) and, of course, the address of function to delete (*proc* parameter.

This function was not documented from WinDom 1.00.

SEE ALSO

EvntDelete().

Programming guideline of WinDom

EvntDisable()

NAME

EvntDisable - Disable all functions binded to an event.

PROTOTYPAGE

int EvntDisable(WINDOW *win, **int** ev);

PARAMETRES

win:
targeted window or NULL,

ev:
event to disable,

DESCRIPTION

EvntDisable() disables an event : functions binded to this event will not be executed (by EvntExec() and by EvntWindom()). This function is used to disabled temporally an event.

SEE ALSO

EvntExec(), EventEnable().

Programming guideline of WinDom

EvntEnable()

NAME

EvntEnable - Enable all functions binded to an event.

PROTOTYPAGE

int EvntEnable(WINDOW *win, **int** ev);

PARAMETRES

win:
targeted window or NULL,

ev:
event to enable,

DESCRIPTION

EvntDisable() enables an event previously disabled by EvntDisable().

SEE ALSO

EvntExec(), EventDisable().

Programming guideline of WinDom

EvntRedraw()

NAME

EvntRedraw - Execute functions binded to **WM_REDRAW** message.

PROTOTYPAGE

void EvntRedraw(WINDOW *win);

PARAMETRES

win:
targeted window.

DESCRIPTION

WM_REDRAW is a special event. When this event occurs, functions binded to this message are executed several times: one time for each rectangle of the AES rectangle list. Off course, EvntWindow() handles that and it is transparent when we write a redraw event function. The consequence is a redraw function cannot be executed directly with EvntExec(). It is the goal of EvntRedraw().

SEE ALSO

EvntExec(), snd_rdw().

Programming guideline of WinDom

snd_rdw()

NAME

snd_rdw - send a redraw message to a window

PROTOTYPAGE

void snd_rdw(WINDOW *win);

PARAMETERS

win: target window.

DESCRIPTION

This function just sends a **WM_REDRAW** message to the work area of a window.

Programming guideline of WinDom

give_iconifyxywh()

NAME

give_iconifyxywh - give a valid position where iconify a window.

PROTOTYPAGE

void give_iconifyxywh(**int** *x, **int** *y, **int** *w, **int** *h);

PARAMETRES

x,y,w,h: position and size of the icon window.

DESCRIPTION

This function can be used if one want iconify artificially a window. The function give valid parameters for WindSet() with the **WF_ICONIFY** mode :

```
{
        int x,y,w,h;
        give_iconify( win, &x, &y, &w, &h);
        WindSet( win, WF_ICONFY, x, y, w, h;
}
```

Depending the OS, the way to compute the iconify position is different:

- with AES older then 4.1, WinDom uses its own icon position. Icon placement is local to the application.
- if cookie 'ICFS' is present, WinDom uses the iconify server to place the icon on the screen, thus icon placement is global.
- with system handling iconify (AES >= 4.1, MagiC), give_iconifyxywh() returns special values (-1 for each component). These values, given to WindSet(WF_ICONIFY), forces the screen manager to place itself the icon.

With PlainTOS, it is not necessary to have an iconify mechanism because EvntWindom() offers to the user to iconify a window when the smaller widget is not present (by shift-clicking the closer widget).

Programming guideline of WinDom

Font library

You should read also the font tutorial section.

FontName2Id()
FontId2Name()
VqtName
VstLoadFonts
VstUnLoadFonts
VstFont
vqt_xname

Programming guideline of WinDom

FontName2Id()

NAME

FontName2Id - convert a font name in a id-font,

PROTOTYPAGE

int FontName2Id(**char** *name);

PARAMETERS

name:
font name,

return:
font id or -1 if font does not exist.

DESCRIPTION

The identificator font is an integer value which identify in a unique way a font. This fonction finds the id-font associated to a font name. The fonction needs a font manager to work correctly.

SEE ALSO

FontId2Name

Programming guideline of WinDom

FontId2Name()

NAME

FontId2Name - convert an id-font in a font name.

PROTOTYPAGE

int FontId2Name(**int** id, **char** *name);

PARAMETERS

id:
font identificator,

name:
name font associated to *id*, it should be a 64-byte buffer.

valeur de retour: -1 si la fonte n'existe pas.

DESCRIPTION

The identificator font is an integer value which identify in a unique way a font. This fonction finds the font name associated to a font identificator. The fonction needs a font manager to work correctly.

SEE ALSO

FontName2Id

Programming guideline of WinDom

VqtName

NAME

VqtName - convert a font name in a id-font,

PROTOTYPAGE

int FontName2Id(**char** *name);

PARAMETERS

name:
font name,

return:
font id or -1 if font does not exist.

DESCRIPTION

VqtName() has the same effect than vqt_name() except it works even if a font driver is not present in memory. The condition is a valid FONTID file exists in the $ETC directory.

SEE ALSO

Programming guideline of WinDom

VstLoadFonts

To Do ...

Programming guideline of WinDom

VstUnLoadFonts

To Do ...

Programming guideline of WinDom

VstFont

To Do ...

Programming guideline of WinDom

vqt_xname

To Do ...

Programming guideline of WinDom

Form library

Programming guideline of WinDom

FormAttach()

NAME

FormAttach() - attach a formular to a window.

PROTOTYPAGE

void FormAttach(WINDOW *win, OBJECT *tree, **void** *func);

PARAMETERS

win:
window descriptor,

tree:
address of object tree or NULL,

func:
address of form evnt function or NULL.

DESCRIPTION

This function is sub function of FormCreate(). It could be used the case where FormCreate() cannot be used. If a **NULL** value is given to parameter *tree*, FormAttach() removes the formalar attached to the window (the formular was prevously attached by FormAttach()).

SEE ALSO

FormCreate().

Programming guideline of WinDom

FormCreate()

NAME

FormCreate - Create a window formular.

PROTOTYPAGE

WINDOW *FormCreate(OBJECT *tree, **int** attrib, **void** (*func)(), **char** *name, GRECT *coord, **int** grow, **int** dup);

PARAMETERS

***tree*:**
address of the object tree,

***attrib*:**
GEM window widgets,

***func*:**
form event function or NULL,

***name*:**
window name,

***coord*:**
position and size of the window or NULL,

***grow*:**
a TRUE value actives the graphic effects when the window is opened and closed,

***dup*:**
a TRUE value duplicates the object tree.

***return*:**
the window descriptor or NULL if error.

DESCRIPTION

FormCreate() creates a window formular and display it to the screen. If a window formular with the same object tree already exists, the function returns the window descriptor associated to this formular. If the window formular is closed, the window is re-open, if the window is iconified, the window is uniconified. The window is eventually topped.

The formular is centered at screen (by using GrectCenter()) if a **NULL** value is given to *coord* parameter. A non **NULL** value of this parameter allows you to define the position in the desktop od the window (coord->g_x and coord->g_y) and the size of the window (coord->g_w and coord->g_h). If a formular is bigger than the window, FormCreate() creates sliders wigdet to make scroll the window. The *name* parameter specifies the window name (equivalent to a WindSet(WF_NAME) call).

It is possible to create several window with a same formular by setting the *dup* parameter of FormAttach() to 1 : the object tree is duplicated in memory (see ObjcDup()) and this copy is used as formular. A such tree is always unique. The raison of using duplicated objects is that each formular has

to have their own coordinates, flags and state. When an object tree is duplicated, the bit **WS_FORMDUP** of the *status* field of the window descriptor is set to one.

FormCreate() attaches to the window specials event functions dedicated to the formular handling (see the TECHNICALS NOTES paragraphe). In perticular, when the user selects an object in a formular, a GEM message is sent. This message has the following structure :

```
evnt.buff[0] = WM_FORM
evnt.buff[1] = GEM application identifior
evnt.buff[2] = 0
evnt.buff[3] = GEM window handle of the formular,
evnt.buff[4] = index of the selected object,
evnt.buff[5] = keyboard state ( a bit field of
                                    K_CTRL, K_ALT, K_LSHIFT, R_SHILT).
```

If an event function was attached to this message (with the parameter *func* of FormCreate() or with EvntAttach()), EvntWindom() will call this function when this event occurs. Instead of using an event function to handle the formular feedback, it is possible to attach variables or function to an object (see ObjcAttach()). A complet example of formular handling can be found in the tutorial The window formulars.

TECHNICAL NOTES

FormCreate() attaches a set of event function dedicated to the formular handling. We list these functions in the following table.

Standard event functions of a formular.

Name	Event	Description
frm_drw	WM_REDRAW	Draw the formular in the window. The
		formular root position is adpated the window
		work area position. Displays the cursor
		in the first text editable object.
frm_dstry	WM_DESTROYED	Close and destroy the window formular.
frm_tpd	WM_TOPPED	Set the window in foreground and eventually
		switch on the cursor of text editable
		object.
frm_mvd	WM_MOVED	Move the window and adaptes the formular
		root position.
frm_keyhd	MU_KEYBD	Handle the keyboard shortcuts and cursor
		movement in editable objects.

SEE ALSO

ObjcChange(), ObjcDraw(), ObjcDup(), ObjcAttach(), FormAttach(), GrectCenter().

Programming guideline of WinDom

FormResize()

NAME

FormResize() - adapte the window size to a formular.

PROTOTYPAGE

void FormResize(WINDOW *win, INT16 *x, INT16 *y, INT16 *w, INT16 *h);

PARAMETERS

win:
window descriptor,

x,y,w,h:
new size and window position (if window is not opened at screen).

DESCRIPTION

This function computes the size of a window formular in order to host the formular. This function can be used to resize a formular when it changes its size.

If the window containing formular is already opened at screen, the window is resize and parameters x, y, w and h have not signification (NULL value can be used). If the window is not opened at screen, x, y, w and h parameters are filled with the new size and position and can be used to call WindSet(WF_CURRXYWH) function.

SEE ALSO

FormCreate(), FormAttach().

Programming guideline of WinDom

FormBegin()

NAME

FormBegin - display a classic formular.

PROTOTYPAGE

void FormBegin(OBJECT *tree, MFDB *bckgrnd);

PARAMETERS

***tree*:**
object tree address of the formular.

***bckgrnd*:**
a pointer to a valid MFDB structure or NULL.

DESCRIPTION

This function creates and displys a classic formular, i.e. a preemptive formular blocking the AES events (not displayed in a window). The formular is centered at screen (by calling the GrectCenter() function). If a **NULL** value is given to parameter *bckgrnd*, the screen area hidden by the formular is not saved in memory. A simple **WM_REDRAW** will be sent when the formular will be closed (FormEnd()). This mode can be used when the formular hides the desktop or windows. If a valid parameter is given to *bckgrnd*, the screen area hidden by the formular will stored in this buffer. This mode should be used when the formular hides another classic formulars.

Correct call of FormBegin():

```
{
        MFDB mem;
        FormBegin( tree, &mem);
}
{
        FormBegin( tree, NULL);
}
```

SEE ALSO

FormDo(), FormEnd(), GrectCenter().

Programming guideline of WinDom

FormDo()

This function is just an alias of the AES form_do() function. Here an example of FormDo() call :

```
{
    MFDB screen;
    OBJECT *tree;
    int res;

    rsrc_gaddr( 0, MY_DIAL, &tree);
    FormBegin( tree, &screen);
    res = FormDo( tree, -1)
    switch(res){
    case OK:
         ...
         break;
    }
    FormEnd( tree, &screen);
}
```

Programming guideline of WinDom

FormEnd()

NAME

FormEnd - close a classic formular.

PROTOTYPAGE

void FormEnd(OBJECT *tree, MFDB *bckgrnd);

PARAMETERS

***tree*:**
object tree address of the formular.

***bckgrnd*:**
a pointer to a valid MFDB structure or NULL.

DESCRIPTION

This function close a formular previously created with FormBegin(). A NULL value of *bckgrnd* parameter sent a **WM_REDRAW** event to the desktop. A correct value of *bckgrnd* restores the screen area hidden by the formular. If the screen area was saved with FormDo() this call releases the memory used.

SEE ALSO

FormDo(), FormBegin().

Programming guideline of WinDom

FormWindBegin()

NAME

FormWindBegin - open a modal window formular.

PROTOTYPAGE

WINDOW *FormWindBegin(OBJECT *dial, char *nom);

PARAMETERS

***dial*:**
address of object tree,

***nom*:**
window title,

***return*:**
window descriptor of the formular.

DESCRIPTION

FormWindBegin() creates a modal window formular that is a form displayed in a modal window. A modal window disables the user interaction of the application. It is very similar to the classic formular except the AES is not stopped. The events of the formular are handled by FormWindDo(). The formular is closed with FormWindEnd().

The user WinDom variable windom.mform.widget defines the widgets of the window.

SEE ALSO

FormWindDo(), FormWindEnd(), windom.mform.widget.

Programming guideline of WinDom

FormWindDo()

NAME

FormWindDo - handle a modal window formular.

PROTOTYPAGE

int FormWindDo(**int** evnt);

PARAMETERS

***evnt*:**
bit field of GEM event to handle,

***return*:**
index of selected object.

DESCRIPTION

This function handles a modal window formular opened by FormWindBegin(). The function returns the index of an **EXIT** or **TOUCHEXIT** object selected by the user. Because this function does not stop the AES events, the parameter *evnt* defines the event to handled. A **MU_MESAG** is always required in order to handled correctly the formular. Now is possible to handle timer or another events.

FormWindDo() uses EvntWindom() to handle events. So if you have binded functions to some events, these events can be handled.

If the **FORM_EVNT** bit of *evnt* parameter is set to one, FormWindDo() returns the last event occured in EvntWindom() (in addition of the selected object index). In this case, the **FORM_EVNT** bit of the returned value is set to one. This feature is now obsolet because WinDom uses a new method to handle GEM events (see EvntAttach()) and all event functions defined by EvntAttach() have a global action.

SEE ALSO

FormWindBegin(), FormWindEnd().

Programming guideline of WinDom

FormWindEnd()

NAME

FormWindEnd - close a modal window formular.

PROTOTYPAGE

void FormWindEnd(**void**);

DESCRIPTION

This function should be call to close a formular opened by FormWindEnd().

SEE ALSO

FormWindOpen(), FormWindDo().

Programming guideline of WinDom

FormSave()

NAME

FormSave - Save a formular state.

PROTOTYPAGE

void FormSave(WINDOW *win, **int** mode);

PARAMETERS

***win*:**
window descriptor,

***mode*:**
OC_FORM or OC_TOOLBAR.

DESCRIPTION

This function saves the state of the objects in a window formular or a toolbar.

EXAMPLE

/* save a formular */

```
{
    OBJECT *tree;
    WINDOW *win;
       static char title[] = "Formular title";
    void (*DoForm)( WINDOW *);  /* see below */

       /* Create a formular */
    rsrc_gaddr( 0, MYDIAL, &tree);
    win = FormCreate( tree, MOVER|NAME|SMALLER|CLOSER,
                            DoForm, title, NULL, TRUE, FALSE);
       /* Save the formular */
    FormSave( win, OC_FORM);
}
```

/* restore the formular (when a CANCEL button is selected) */

```
void GereForm( WINDOW *win)
{
     if( evnt.buff[4] == MYDIAL_ANNUL)
     {
          Restore( win, OC_FORM);  /* On restore l'état */
          snd_msg( win, WM_CLOSED);
     }
}
```

REMARKS

Memory used by FormSave() is free up when the window is destroyed by the standard form event function in a case of window formular or by WindDelete() and WindSet(, WF_TOOLBAR, NULL) in a case of a toolbar.

SEE ALSO

FormRestore().

Programming guideline of WinDom

FormRestore()

NAME

FormRestore - Restore a formular state.

PROTOTYPAGE

void FormRestore(WINDOW *win, **int** mode);

PARAMETERS

***win*:**
window descriptor,

***mode*:**
OC_FORM (formular) or OC_TOOLBAR (toolbar).

DESCRIPTION

This function restores the objects state of a formular or a toolbar previously saved by FormSave().

SEE ALSO

FormSave().

Programming guideline of WinDom

FormAlert()

NAME

FormAlert() - display a GEM alert box.

PROTOTYPAGE

void FormAttach(**int** but, **char** *msg, ...);

PARAMETERS

see form_alert()

DESCRIPTION

This function is a like the GEM function form_alert() execept, it is possible to print variables as vprintf().

EXAMPLE

```
FormAlert( 1, "[1][Mesag %d occurs.][OK]", evnt.buff[0]);
```

SEE ALSO

FormCreate().

Programming guideline of WinDom

FormThumb()

NAME

FormThumb() - handle a thumb index in a dialog box.

PROTOTYPAGE

int FormThumb(WINDOW *win, **int** *idxthb, **int** *idxbut, **int** nb)

PARAMETERS

***win*:**
window descriptor,

***idxthb*:**
list of subdialog index to link ,

***idxbut*:**
list of button index to link,

***nb*:**
number of buttons to link,

return:
0 if no error occurs.

DESCRIPTION

The function declare a thumb, i.e. a multiple subdialog, inside a window dialog. After this call, thumb buttons are automatically handled. Parameters *idxthb* and *idxbut* describe links between thumb buttons and sub dialogs. There are array of *nb* items. For each item i of these arrays, idxthb[i] is linked to idxbut[i].

EXAMPLE

```
/* Example of thumb indexes with three elements */
int but [] = {BUT1, BUT2, BUT3};
int sub [] = {SUB1, SUB2, SUB3};

win = FormCreate( ...);
FormThumb( win, sub, but, 3);
```

SEE ALSO

FormCreate().

Programming guideline of WinDom

FormThbSet()

NAME

FormThbSet() - change active thumb.

PROTOTYPAGE

void FormThbSet(WINDOW *win, **int** but);

PARAMETERS

***win*:**
window descriptor,

***but*:**
button index linked to thumb to active,

DESCRIPTION

This function changes the active thumb (without a user manipulation).

SEE ALSO

FormThumb(), FormThbGet().

Programming guideline of WinDom

FormThbGet()

NAME

FormThbSet() - returns active thumb.

PROTOTYPAGE

int FormThbSet(WINDOW *win, **int** mode);

PARAMETERS

***win*:**
window descriptor,

***mode*:**
0 : return active button ; 1 : return active thumb. 'item [return:] object index or -1 if error.

DESCRIPTION

This function returns the active thumb (the thumb itself or button linked to the active thumb).

SEE ALSO

FormThumb(), FormThbSet().

Programming guideline of WinDom

Frame library

Programming guideline of WinDom

FrameInit()

NAME

FrameInit() - Initialization of the frame environment.

PROTOTYPAGE

void FrameInit(void);

DESCRIPTION

This function initializes the frame environment. When the frame environ is not used, the code size is smaller. From WinDom version of January 1998, the function is required. The frame environ should be released with FrameExit().

SEE ALSO

FrameExit().

Programming guideline of WinDom

FrameExit()

NAME

FrameExit() - release the frame environment.

PROTOTYPAGE

void FrameExit(void);

DESCRIPTION

This function release the memory used by the frame environment.

SEE ALSO

FrameInit().

Programming guideline of WinDom

FrameCreate()

NAME

FrameCreate() - Create a frame window.

PROTOTYPAGE

WINDOW *FrameCreate(int attrib);

PARAMETERS

item:
GEM widget of the window,

return:
window descriptor.

DESCRIPTION

FrameCreate() just creates a window descriptor whose will host framed windows. The window created is not opened at screen as for WindCreate(). Optional parameters can be set with WindSet() for general window options and FrameSet() for specific frame options.

The window created by FrameCreate() has the **WS_FRAME_ROOT** bit of the *status* window descriptor field set to 1. A structure FRAME structure is attached as data to the window with a **WD_WFRA** magic number.

The method used to handle frame window is very simple. A frame window is seen like a set of framed windows. A framed window is not a real GEM window but is identified by a window descriptor. The frame window, i.e. the root window holding the list of framed window, and the standard event functions of the frame window use the standard event function of the framed window. Some events are applied to the active frame, such as a button event. The active frame may be defined by FrameSet().

BUGS

- sliders of framed window are not correctly initialized,
- the vertical frame resizing widget are not supported.

SEE ALSO

FrameAttach(), FrameSet().

Programming guideline of WinDom

FrameAttach()

NAME

FrameAttach() - attach a framed window in a window.

PROTOTYPAGE

void FrameAttach(WINDOW *win, WINDOW *frame, **int** line, **int** col, **int** w, **int** h, **int** flags);

PARAMETERS

win:
window descriptor of the root window,

frame:
window descriptor of the window to attach,

line, col:
cell which will contain the frame,

w, h:
size of the frame,

flags:
bit field of special frame features (see FramSet()).

DESCRIPTION

This function attaches a window in a root window. The attached window becomes a framed window. The framed window is created like a standard window. It can be a formular or any custom user window. The window descriptor of the framed window is removed from the list of window and the **WS_FRAME** bit of the window descriptor *status* field is set to 1.

Frames inside a window are organized by line of cells. Each line can contain variable cells. The *line* and *col* parameters defines the order of the frame in the window. The real coordinates of a frame inside the work area of the root window depend on the frame sizes. And these sizes can be eventually changed by the user. The *w* and *h* parameters define the initial size of the frame.

The *flags* parameter defines special feature of the frame. See FrameSet() for a complet description of this parameter.

The frame uses the GEM widget of the window. Possible attributs for a frame are **INFO**, **SIZER**, **HSLIDE**, **VSLIDE**, **UPARROW**, **DNARROW**, **LFARROW** and **RTARROW**. Other widgets are ignored.

SEE ALSO

FrameCreate(), FrameSet(), FrameRemove().

Programming guideline of WinDom

FrameRemove()

NAME

FrameRemove() - Remove a framed window from a root window.

PROTOTYPAGE

WINDOW *FrameRemove(WINDOW *win, WINDOW *frame, **int** line, **int** col);

PARAMETERS

win:
root window descriptor,

frame:
framed window descriptor or NULL,

line, col:
cell coordinate of the framed window,

return:
window descriptor of the frame removed.

DESCRIPTION

FrameRemove() removes a framed window from a root window. The removed frame becomes a normal window. If the *frame* parameter has a **NULL** value, the *line* and *col* cell coordinates are used to locate the framed window to remove.

SEE ALSO

FrameAttach().

Programming guideline of WinDom

FrameSet()

NAME

FrameSet() - Frame settings.

PROTOTYPAGE

void FrameSet(WINDOW *win, **int** mode, ...);

PARAMETERS

win:
window descriptor (of a root frame window of a framed window),

mode:
parameter to set,

...:
new values depending on *mode*.

DESCRIPTION

This function sets the special features of frame windows. The table below lists and comments the different modes of the function.

FrameSet() mode

Mode	**Type window**	**Parameters**	**Remarks**
FRAME_BORDER	root window	set the border size of frames	pixel unit
		FrameSet(..,border);	
FRAME_COLOR	root window	set the border frame color	VDI color index
		FrameSet(..,color);	
FRAME_KEYBD	root window	define the frame catching an	
		keyboard event.	
		The frame pointed by mouse:	
		FrameSet(.., MOUSE_WINDOW);	
		The active frame:	
		FrameSet(.., FRONT_WINDOW);	
FRAME_ACTIVE	root window	Makes a frame active :	frame is a WINDOW
		FrameSet(..,frame);	pointer.
FRAME_TOPP-	root window	A topped frame becomes active	A TRUE or FALSE
ED_ACTIV		FrameSet(..,TRUE);	value.
FRAME_SIZE	framed window	set the framed window size	w and h units depends
		FrameSet(..,w,h);	on FRAME_FLAGS values.
FRAME_FLAGS	framed window	set a flag:	flag values are
		FrameSet(..,flag,TRUE)	listed in the next
		unset a flag:	table.
		FrameSet(..,flag,TRUE)	

Frame flags (related to FRAME_FLAGS mode)

FLAGS	**Descriptions**
FRAME_HSCALE	The framed window height is defined
	proportionaly to the window root
	work area height (0..100).
FRAME_WSCALE	The framed window width is defined
	proportionaly to the window root
	work area width (0..100).
FRAME_HFIX	The framed window height is an absolute
	value (in pixel).
FRAME_WFIX	The framed window width is an absolute
	value (in pixel).
FRAME_NOBORDER	The framed window has no borders.
FRAME_SELECT	The framed windows is activable.

Programming guideline of WinDom

FrameGet()

NAME

FrameGet() - get frame related informations.

PROTOTYPAGE

void FrameGet(WINDOW *win, **int** mode, ...);

PARAMETERS

win:
window descriptor (of a root frame window of a framed window),

mode:
parameter to get,

...:
new values depending on *mode*.

DESCRIPTION

From WinDom version 1.20, frame sub structures in window descriptor have been hidden to improve portability with futur extensions. To compense the loss of information, the function FrameGet() has been introduced. Currently, only one mode is available.

FrameGet() mode

Mode	Type window	Description/Parameters
FRAME_CELL	framed window	returns cellule reference
		par1 : line position
		par2 : row position
	root window	Currently, no mode available.

Programming guideline of WinDom

FrameFind()

NAME

FrameFind() - find a framed window.

PROTOTYPAGE

WINDOW *FrameFind(WINDOW *win, **int** x, **int** y);

PARAMETERS

win:
root frame window descriptor,

x, y:
coordinate in desktop,

return:
framed window descriptor found.

DESCRIPTION

FrameFind() finds the framed window pointed by the *x*, *y* coordinates in the *win* root frame window. A **NULL** value returned indicates the frame is not found.

SEE ALSO

FrameSearch().

Programming guideline of WinDom

FrameSearch()

NAME

FrameSearch() - Find a frame by cell reference.

PROTOTYPAGE

WINDOW *FrameSearch(WINDOW *win, **int** line, **int** col);

PARAMETERS

win:
frame root window descriptor,

line, col:
framed window cell coordinates,

return:
framed window descripteur found.

DESCRIPTION

Returns the framed window descriptor of the *line*, *col* cell in the *win* root frame window. A **NULL** value returned indicates the frame is not found.

SEE ALSO

FrameFind().

Programming guideline of WinDom

FrameCalc()

NAME

FrameCalc() - get the framed window work area.

PROTOTYPAGE

int FrameCalc(WINDOW *win, **int** mode, INT16 *x, INT16 *y, INT16 *w, INT16 *h);

PARAMETERS

win:
framed window descriptor,

mode:
zone courante (1) ou zone de travail (0),

x,y,w,h:
position and size of the work area,

return:
a nul value if no error.

DESCRIPTION

FrameCalc() computes the coordinates and size of the work area of a framed window. Widgets are take in consideration. This function is a sub-function of WindGet(). In fact, WindGet(WF_WORKXYWH) works correctly with framed window. The **WF_CURRXYWH** mode is supported too. It better to use WindGet(), specially in event functions because these windows can be used as normal window or as framed window.

SEE ALSO

WindGet().

Programming guideline of WinDom

WindSet()/WindGet() and frames

There is an important remark to do here. When you use WindSet() and WindGet() with the mode **WF_WORKXYWH** on a framed window descriptor, these function compute the frame work area size and not the window area size. Thus, WindGet(..., WF_WORKXYWH, ...) is identical to FrameCalc(). This feature allows us to use directly event function initially written for window. So, prefined window, such as dialog box, can be framed.

Programming guideline of WinDom

Selectors library

Programming guideline of WinDom

FselInput()

NAME

FselInput - universal file selector.

PROTOTYPAGE

int FselInput(**char** *path, **char** *name, **char** *ext, **char** *title **char** *lpath, **char** *lext)

PARAMETERS

path:
directory where the file selector is opened, then the directory of the selected item.

name:
name of a default file, then the name of the selected file,

ext:
file mask,

title:
selector title,

lpath:
list of predefined directores or NULL,

lext:
list of predfined file mask of NULL,

return:
1 if an object has been selected, 0 else.

DESCRIPTION

FselInput() is a custom call of the GEM file selector. If an alternative file selector is available ('FSEL' cookie, Selectric, BoxKite 2, FLSX extensions), it is used instead of the GEM standard file selector.

When the function returns, parameters *path* and *name* are filled in with the directory and the filename selected. Then FselInput() can be used as a file and a directory selector. If the *path* parameter is an empty string, the current directory is used. If the *ext* parameter is an empty string, the default mask used by FselInput() will be "*.*".

Parameter *title* set fileselector title and can be used for any TOS version. If fileselelector does not support this feature, that parameter has not effect.

The *lpath* and *lext* allows you to define a list of preset directories. There list are displayed inside the file selector if it is possible that is the case with Selectrics, BoxKit 2 and the FLSX selectors. A list is a string whose each items are delimited by a ; character:

```
"C:\\\USR;C:\\USR\BIN"
"*.C,*.H;*.PRJ,*.RSC"
```

FselInput() adds in the directories preset, the path of the user directory (if the $HOME environ variable is defined). The environ variables $FSELPATH and $FSELMASK are used to build a default list of directories and file mask. The additional lists given by *lpath* and *lext* are adding in these lists. Environ variables provide a way to configure globally all WinDom clients. In addition, it is possible, for the user, to configure a specific application : the windom.fsel.path and windom.fsel.mask variable in the configuration file define the list of directories and file mask. These values are adding to the lists displayed by the file selector.

If the system has a FSLX extension (see appl_getinfo()), the file selector is display in a modal window.

The windom.fsel.fslx variable from the WinDom configuration file set to FALSE forces FselInput() to not use the FSLX extension.

```
int CallFsel ( name) {
       static char path[255]=""; /* Fist usage : current directory */
    char fullname[255]="";

    if( FselInput( path, name, "Load a file", "", NULL, NULL)) {
        strpcy( completname, path);
        strcat( completname, name);
        strpcy( name, fullname); /* return the full path */
        return 1;
    } else
                return 0;
 }
```

SEE ALSO

windom.fsel.path, windom.fsel.mask, windom.fsel.fslx

Programming guideline of WinDom

FontSel()

NAME

FontSel - Font selector

PROTOTYPAGE

int FontSel(**char** *winname, **char** *example, **int** flags, **int** *fontid, **int** *fontsize, **char** *fontname);

PARAMETERS

winname:
selector title,

example:
text used to display the font or NULL,

flags:
bit field:

VSTHEIGHT:
size in pixel unit (instead of point unit),

MONOSPACED:
use only non proportional fonts.

fontid:
identificator of the selected font,

fontsize:
size selected,

fontname:
font name filled in a 64-byte buffer,

return:
1 if the user choice is valid, 0 else.

DESCRIPTION

FontSel() calls the internal WinDom font selector. The selector is displayed in a modal window. The selector can be used even the system does not support multiple fonts, in this case only the size can be changed.

Before the call, the *fontid* and *fontsize* parameters may be filled with a default font-id and size. If a null value is used, default values are the system font and a size of 13 pixels. If the

example parameter is NULL, a default text is used to display the fonts.

The window hosting fontselector catches the **AP_TERM** message : the function will terminate in the same manner than a user clic on the Cancel button.

USAGE OF SELECTOR

simple clic
select a font and display it,

double clique
select a font and return this selection, the selector is closed,

size
font size are real time updated,

up and down arrow
select a font,

touche return ou enter
keyboard shortcut of the OK button,

touche Undo
keyboard shortcut of the CANCEL button,

Programming guideline of WinDom

Inquire library

This library contains useful functions for testing some system feature.

Programming guideline of WinDom

has_appl_getinfo()

NOM

has_appl_getinfo() - test if the appl_getinfo() function is avalaible.

PROTOTYPAGE

int has_appl_getinfo(**void**);

DESCRIPTION

This function returns 1 if appl_getinfo() is present. It is the case with :

- MultiTOS 1 and 1.2 (AES 4 and AES 4.1)
- Naes
- Geneva
- MagiC 3
- Wdialog
- WinX 2.1
- when the call appl_find("?AGI") returns a value different of -1.

With non multitasking TOS, it is a good idea to use WinX or/and Wdialog.

VOIR AUSSI

appl_getinfo()

Programming guideline of WinDom

vq_gdos()

NOM

vq_gdos - test if GDOS or equivalent is available.

PROTOTYPAGE

int vq_gdos(**void**);

Programming guideline of WinDom

vq_vgdos()

NOM

vq_vgdos - test if SpeedoGdos or equivalent is available.

PROTOTYPAGE

int vq_vgdos(void);

NOTES

NVDI from version 3 is SpeedoGDOS compatible.

Programming guideline of WinDom

vq_magx()

NOM

vq_magx - test if MagiC is present and returns the version number.

PROTOTYPAGE

int vq_magx(void);

DESCRIPTION

The function returns the MagiC version number (from MagiC 3) or 0 if MagiC is not present.

Programming guideline of WinDom

vq_tos()

NOM

vq_tos - returns the TOS version number.

PROTOTYPAGE

int vq_tos(void);

SEE ALSO

The AES versions annexe.

Programming guideline of WinDom

vq_naes()

NOM

vq_naes - returns the Naes version number.

PROTOTYPAGE

int vq_naes(void);

DESCRIPTION

vq_naes() returns the Naes version number if present or 0 else.

Programming guideline of WinDom

vq_nvdi()

NOM

vq_nvdi - returns the Nvdi version number.

PROTOTYPAGE

int vq_nvdi(void);

DESCRIPTION

vq_nvdi() returns the Nvdi version number if present or 0 else.

VOIR AUSSI

vq_gdos(), vq_gvdos()

Programming guideline of WinDom

vq_winx()

NOM

vq_winx - returns the WinX version number.

PROTOTYPAGE

int vq_winx(void);

DESCRIPTION

vq_winx() returns the WinX version number or zero if WinX is not installed. The version number is available from WinX 2.1. From older version, vq_winx() returns always 0x0100.

Programming guideline of WinDom

vq_extfs()

NAME

vq_extfs - test of long filename.

PROTOTYPAGE

int *vq_extfs(**char** *path)

PARAMETERS

path:
complet path of a file or folder,

retour:
1 if the file system supports long file name, 0 else.

DESCRIPTION

vq_extfs() tests if the file system supports long file name. MiNT and MagiC are concerned.

Programming guideline of WinDom

Menu library

Programming guideline of WinDom

MenuBar()

NAME

MenuBar - defines the desktop menu.

PROTOTYPAGE

int MenuBar(OBJECT *menu, **int** mode);

PARAMETRES

menu:
address of the menu objet tree,

mode:
1 displays the menu, 0 removes the menu,

return:
0 if no error.

DESCRIPTION

MenuBar() remplaces the AES menu_bar() function. menu_bar() should never used in the WinDom environment.

VOIR AUSSI

MenuTnormal(), MenuIcheck(), MenuText().

Programming guideline of WinDom

MenuTnormal()

NAME

MenuTnormal - hilight an entry in a menu.

PROTOTYPAGE

int MenuTnormal(WINDOW *win, **int** title, **int** mode);

PARAMETERS

win:
window descriptor or NULL,

title:
index of the entry in the menu,

mode:
0 hilights the entry and 1 unhilights it,

return:
0 if no error.

DESCRIPTION

MenuTnormal() replaces the AES menu_tnormal() function. It can be used on window menu or on the desktop menu (win=NULL). MenuTnormal() it usually used to unlight a menu entry when a **MN_SELECTED** or **WM_MNSELECTED** message occurs.

SEE ALSO

MenuBar()

Programming guideline of WinDom

MenuIcheck()

NAME

MenuIcheck - Check/uncheck an entry in a menu.

PROTOTYPAGE

int MenuIcheck(WINDOW *win, **int** index, **int** mode);

PARAMETERS

win:
window descriptor or NULL,

index:
item index in the menu,

mode:
1 checks the item and 0 unckecks the item.

return:
0 if no error.

DESCRIPTION

MenuIcheck() is the remplacent of the AES menu_icheck() function. It is used to check or uncheck an item in a window menu or the desktop menu (win=NULL).

SEE ALSO

MenuBar()

Programming guideline of WinDom

MenuText()

NAME

MenuText - change the text of a menu item.

PROTOTYPAGE

int MenuText(WINDOW *win, **int** index, **char** *txt);

PARAMETERS

win:
window descriptor or NULL,

index:
index of the menu item,

txt:
new text,

return:
0 if no error.

DESCRIPTION

MenuText() is the WinDom eqivalent of the AES menu_text() function. It is used to change the text of an item in a window menu or the desktop menu. MenuText() should be always used instead of ObjcString().

SEE ALSO

MenuBar()

Programming guideline of WinDom

MenuDisable()

NAME

MenuDisable - Disable the desktop menu.

PROTOTYPAGE

int MenuDisable(**void**);

PARAMETERS

return: 0 if no error.

DESCRIPTION

MenuDisable() disables the desktop menu: the user cannot select items except the desktop accessories. This function is used by FormWindBegin(), the file and font selectors.

SEE ALSO

MenuEnable().

Programming guideline of WinDom

MenuEnable()

NAME

MenuEnable - Enable the desktop menu.

PROTOTYPAGE

int MenuEnable(**void**);

PARAMETERS

return: 0 if no erreur.

DESCRIPTION

MenuEnable() enables the desktop menu previously disabled by MenuDisable(). This function is used by FormWindEnd(), the file and font selectors.

SEE ALSO

MenuDisable().

Programming guideline of WinDom

MenuPopUp()

NAME

MenuPopUp - Display and handle a menu popup.

PROTOTYPAGE

int MenuPopUp(void *data, **int** xpos, **int** ypos, **int** size, **int** seen, **int** item, **int** mode);

PARAMETERS

data:
address of a valid object tree or a list of entries (see **P_LIST** mode),

xpos, ypos:
menu popup position in the desktop,

size:
if **P_LIST** is used, indicates the number of entries *data*,

seen:
if **P_LIST** is used, indicated the maximum of entries seen in the popup.

item:
the popup position is adjusted in order to match the *item* entry with the *x*, *y* coordinates,

mode:
a bit field on:

P_RDRW
this bit means that a simple redraw message will be sent to screen to redraw the area hiding by the popup instead of save the area in a buffer. This mode can be used if the popup is called from a window. With the **P_WNDW** mode, this mode is always used,

P_WNDW
the popup will be drawn in a window instead of a classic formular. This mode allows to not stop the AES events. It should not be used when the popup is call from a classic formular,

P_LIST
This mode means that *data* parameter points to list of entries. A list of entries is a array of string. Each string is a label in the popup menu. It allows to creating popup without use object tree.

P_CHCK

The *item* entry will be checked.

return:

index of item selected in the popup or -1 if no selection.

DESCRIPTION

If *P_WNDW* mode is set, it can be disabled by the user if the windom.popup.window variable in the WinDom configuration file is set to FALSE. This mode should be used always be used when it is possible (a call of MenuPopup over a window) because the user has the choice to enable or disable this feature.

A list of entries is a pointer such as "char *ptxt[]". If the *seen* parameter is used, the popup is displayed with a slider and contains *seen* items.

Keyboards can be used to naviguate in the popup (up and down arrow) and validate an entry (RETURN or ENTER keys).

If you use mode **P_WNDW**, MenuPopUp() can be displayed in a window. This mode allows MenuPopup() to not stop AES events. It is very usefull with a multitasking system. If you call MenuPopUp() form a classic dialog box, this mode should never be used. In other case, this mode should always be used because it can be disabled or enabled by the user using the WinDom configuration file (see windom.popup.window variable).

With a **P_LIST** mode, the menu popup look can be defined by the user from the WinDom configuration file (see windom.popup variables).

The **P_RGHT** mode is now obsolet.

SEE ALSO

windom.popup

Programming guideline of WinDom

MenuScroll()

NAME

MenuScroll - scroll the entries of a window menu.

PROTOTYPAGE

void MenuScroll (WINDOW *win, **int** dir);

PARAMETERS

win:
window descriptor,

dir:
1 - left direction
0 - right direction

DESCRIPTION

This function scrolls the entries (titles) of a menu in a window. Notice if the variable windom.menu.scroll in the WinDom configuration file is set to 1, a scroller widget appears in the window menu bar: the user can himself scrolls the menu.

SEE ALSO

windom.menu.scroll

Programming guideline of WinDom

Mouse Library

Programming guideline of WinDom

MouseObjc()

NAME

MouseObjc - center the mouse sprite on an object.

PROTOTYPAGE

void MouseObjc(OBJECT *tree, **int** index);

PARAMETERS

tree:
object tree address,

index:
targeted object index.

DESCRIPTION

MouseObjc() sets the mouse sprite at the center of an object in a formular of toolbar.

Programming guideline of WinDom

MouseSprite()

NAME

MouseSprite - set the mouse sprite.

PROTOTYPAGE

void MouseSprite(OBJECT *tree, **int** index);

PARAMETERS

tree:
object tree address,

index:
targeted object index.

DESCRIPTION

MouseSprite() defined the sprite mouse. The sprite mouse is defined by a monochrome icon from a resource. Background and foreground color of icon are applied to the mouse sprite.

SEE ALSO

graf_mouse()

Programming guideline of WinDom

MouseWork()

NAME

MouseWork - mouse sprite animation.

PROTOTYPAGE

void MouseWork(**void**);

DESCRIPTION

MouseWork() changes the mouse sprite in order to give an impression of animation : the mouse looks like a turning disc. You have to call severals times the function to give an animation effect.

EXAMPLE

```
/* perform a background work ... *//
while( !end) {
        MouseWork();
        end = process();
        EvntWindom( MU_MESAG);
}
graf_mouse( ARROW, 0L);
```

SEE ALSO

graf_mouse(), MouseSprite()

Programming guideline of WinDom

Object library

Programming guideline of WinDom

ObjcAttach()

NAME

ObjcAttach() - attach a variable or a function at an object.

PROTOTYPAGE

int ObjcAttach(**int** mode, **WINDOW** *win, **int** index, **int** type, **void** *data, ...);

PARAMETERS

***mode*:**

OC_FORM:
if the object is in a window formular,

OC_TOOLBAR:
object is in a toolbar,

OC_MENU:
object is in a window menu,

***win*:**
a window descriptor or NULL,

***index*:**
object index to attach,

***type*:**

BIND_VAR:
attach a variable,

BIND_BIT:
attach a specific bit of a variable,

BIND_FUNC:
attach a function.

***data*:**
data or function address to attach,

...:
additional parameter depending on *type* parameter : in **BIND_BIT** mode, this parameter specifies the bit to attach in *data*. In **BIND_FUNC** mode, this parameter can specify an optional user data pointer which pass to the binded function.

***return*:**

a negative code error.

DESCRIPTION

ObjcAttach() attaches a function or a variable at an object from a window formular, a toolbar or a menu. The rules are differents if you use a formular and a toolbar or a menu.

With formulars or toolbars, only **EXIT** or **TOUCHEXIT** objects can be attached at a function. When the user selects these objects, the function is invoked. Only **SELECTABLE** objects can be attached at a variable. If the object has the **RADIO** flag, the variable attached - always an integer variable - is filled with the index of the selected RADIO object at the RADIO level. If the object is not a RADIO object, the variable is filled with 1 if the object is selected, 0 else with the **BIND_VAR** mode. The **BIND_BIT** mode allows the object to be attached with a specific bit of the variable. This bit is specified by the *bit* parameter.

With menu, an item of the menu can be attached at a function or at a variable. When an item is linked to a variable, it is checked or unchecked when the user selects it. The variable linked is filled with 1 or 0 (or a specific bit with the **BIND_BIT** mode) when the item is checked or unchecked. Notice desktop menu is addressed if win *parameter* is set to **NULL**.

A function linked with to an object in a formular or a toolbar has the following interface:

```
void func ( WINDOW *win, int index, int mode, void *data);
```

where *win* is the host window, *index* is the index of the attached object and *mode* can be **OC_FORM**, **OC_TOOLBAR** or **OC_MENU**. If an user data pointer is specified with ObjcAttach(), this pointer can be read as a fourth parameter of the binded function (*data* in our example).

A function linked to a menu object has an additionnal parameter - *title* - which indicated the menu title index selected. This parameter is required by MenuTNormal():

```
void func ( WINDOW *win, int index, int mode, int title, void *data);
```

The fifth parameter *data* is an optional user pointer data specified in ObjcAttach().

EXAMPLES

```
   {
          static int radio = RAD1;
#define OPTION1    0x1     /* bit 0 */
#define OPTION2 0x2        /* bit 1 */
#define OPTION3 0x4 /* bit 2 */
          static int options = 0;

          /* Before : create the form with FormCreate() */
          /* Then attach the objects  */

          /* 3 radio buttons in a formular */
          ObjcAttach( OC_FORM, win, RAD1, BIND_VAR, &radio, 0);
          ObjcAttach( OC_FORM, win, RAD2, BIND_VAR, &radio, 0);
          ObjcAttach( OC_FORM, win, RAD3, BIND_VAR, &radio, 0);

          /* some checkboxes ... */
          ObjcAttach( OC_FORM, win, BUT1, BIND_BIT, &options, OPTION1);
          ObjcAttach( OC_FORM, win, BUT2, BIND_BIT, &options, OPTION2);
          ObjcAttach( OC_FORM, win, BUT3, BIND_BIT, &options, OPTION3);

          /* An example of function linked to an object   *
           * see after for the definition of the function */
          ObjcAttach( OC_FORM, win, OK, BIND_FUNC, ButOk, 0);
```

```
    }

    /* Function linked to OK object */
    void ButOk( WINDOW *win, int index, int mode) {
            /* Unselect the object ... */
            ObjcChange( mode, win, index, NORMAL, 0);
            /* ... and destroy the window */
            ApplWrite( app.id, WM_DESTROY, win->handle);
    }
}
```

Programming guideline of WinDom

ObjcDraw()

NAME

ObjcDraw() - draws an objet in a formular.

PROTOTYPAGE

int ObjcDraw(**int** mode, **void** *win, **int** index, **int** depth);

PARAMETERS

***mode*:**

OC_FORM:
if the formular is a window,

OC_TOOLBAR:
if the formular is a toolbar,

OC_OBJC:
if is a classic formular,

***win*:**
a window descriptor or an object tree (OC_OBJC),

***index*:**
object index to draw,

***depth*:**
depth,

return:
a negative code error.

DESCRIPTION

This function replaces the AES objc_draw() function. It is specially designed to draw object in window formular (**OC_FORM** mode) or toolbar (**OC_TOOLBAR** mode) but it works on classical formular (**OC_OBJC** mode) too. Using this last mode, ObjcDraw() is equivalent to objc_draw(). When ObjcDraw() works on a window, the redraw is done with repect to the window clipping. It works even the window formular is behind an another window. If the *depth* parameter has its **OC_MSG** sets to 1, the object will be drawn by sending a set of **WM_REDRAW** messages to the AES kernel (the draw will be handle by EvntWindom()) instead of draw immediatly the object.

WARNING

This function should never be used in a window redraw function (i.e. a function associated to a **WM_REDRAW** message) because this function is invoked by EvntWindom() on each rectangle of the AES rectangle list and the ObjcDraw() function uses this list too. If you really want draw an object prefer a window formular or use objc_draw() and the global variable *clip* as clipping area:

```
objc_draw( tree, ROOT, MAXDEPTH, clip.g_x, clip.g_h, clip.g_w, clip.g_h);
```

SEE ALSO

ObjcChange(), objc_draw()

Programming guideline of WinDom

ObjcChange()

NAME

ObjcChange() - change the object state.

PROTOTYPAGE

int ObjcChange(**int** mode, **void** *win, **int** index, **int** state, **int** redraw);

PARAMETERS

***mode*:**

OC_FORM:
if the formular is a window,

OC_TOOLBAR:
if the formular is a toolbar,

OC_OBJC:
if is a classic formular,

***win*:**
a window descriptor or an object tree (OC_OBJC),

***index*:**
object index to draw,

***state*:**
new object state,

***redraw*:**
if different to zero, the object is redrawn,

***return*:**
a negative code error.

DESCRIPTION

This function replace the AES objc_change() function mainfully to change the object state in window formular or toolbar. If you use ObjcChange() with the **NORMAL** value, the extended bits of the objet state (known as extended states) will be not affected. Negative states (e.g. SELECTED) have effect to unset the state. See ObjcDraw() for recommandation usage.

SEE ALSO

ObjcDraw().

Programming guideline of WinDom

ObjcEdit()

NAME

ObjcEdit() - control texte edition in EDITABLE object.

PROTOTYPAGE

int ObjcEdit(**int** mode, **void** *win, **int** obj, **int** val, **INT16** *idx, **int** kind);

PARAMETERS

***mode*:**

OC_FORM:
if the formular is a window,

OC_TOOLBAR:
if the formular is a toolbar,

OC_OBJC:
if is a classic formular,

***win*:**
a window descriptor or an object tree (OC_OBJC),

***obj*:**
object index to edit,

***val*:**
parameter depending on the *kind* parameter,

****idx*:**
position of cursor,

***kind*:**
possible values are :

ED_INIT
activate the cursor initiale position is given by *idx,

ED_END
desactivate the cursor

ED_CHAR
insert the charater 'val' at current cursor position,

ED_BLC_OFF

desactivate a selection,

ED_BLC_START
set the beginning of a selection,

ED_BLC_END
set the end of a selection activate it,

***return*:**
a negative code error.

DESCRIPTION

This function replaces the AES objc_edit() function and works with all editable field. ED_BLC_OFF, ED_BLC_START and ED_BLC_END are extended modes only valid with XEDIT objects (WinDom extended editable fields). This function is currently under developpement.

SEE ALSO

objc_edit()

Programming guideline of WinDom

ObjcWindDraw()

NAME

ObjcWindDraw - draw any object in any window.

PROTOTYPAGE

int ObjcWindDraw(WINDOW *win, OBJECT *tree, **int** index, **int** depth, **int** xclip, **int** yclip, **int** wclip, **int** hclip);

PARAMETERS

***win*:**
host window,

***tree*:**
address of object tree,

***index*:**
object index to draw,

***depth*:**
depth,

***xclip*, *yclip*, *wclip*, *hclip*:**
clipping area,

***return*:**
a negative code error.

DESCRIPTION

This function is a sub-function of ObjcDraw(). It allows you to draw an object tree in a window with repect to the window workspace. As ObjcDraw(), this function should never be used in a window redraw function.

SEE ALSO

ObjcDraw(), ObjcWindChange()

Programming guideline of WinDom

ObjcWindChange()

NAME

ObjcWindChange - change the state of any object tree in any window.

PROTOTYPAGE

int ObjcWindChange(WINDOW *win, OBJECT *tree, **int** index, **int** xclip, **int** yclip, **int** wclip, **int** hclip, **int** state);

PARAMETERS

***win*:**
host window,

***tree*:**
address of object tree,

***index*:**
object index to change,

***xclip*, *yclip*, *wclip*, *hclip*:**
clipping area,

***state*:**
new object state,

***return*:**
a negative code error.

DESCRIPTION

This function is a sub-function of ObjcChange(). See ObjcWindDraw() and ObjcChange() for full details.

SEE ALSO

ObjcChange(), ObjcWindDraw(), ObjcDraw()

Programming guideline of WinDom

ObjcDup()

NAME

ObjcDup - objects duplication.

PROTOTYPAGE

OBJECT *ObjcDup(OBJECT *tree, WINDOW *win);

PARAMETERS

***tree*:**
address of object tree to duplicate,

win:
window descriptor if form is hosted in a window,

***return*:**
address object tree duplicated.

DESCRIPTION

ObjcDup() performs a dynamic copy of an object tree. The new object tree created is different but have the same properties. Currently, string, image, icon and editable field are not duplicated but it should do (project). An object tree created with ObjcDup() should be free by ObjcFree().

If object tree contains **USERDRAW** object, the parameter *win* is absolutely required. In other case, **NULL** is a correct value;

This function is used by FormCreate() to open several formular with the same object tree. If the **WS_FORMDUP** bit of the *status* window descriptor field is set to 1, the standard destruction function release the memory with ObjcFree().

FormAttach() does not duplicated the object tree. If you create multiple window formular from an unique object tree with FormAttach() you should duplicate the object tree with ObjcDup().

Toolbars and menus attached to a window with WindSet() are duplicated in memory using ObjcDup() and the memory automatically released when the window is destroyed.

SEE ALSO

ObjcFree(), FormCreate().

Programming guideline of WinDom

ObjcFree()

NAME

ObjcFree - release a duplicated object tree.

PROTOTYPAGE

void ObjcFree(OBJECT *tree);

PARAMETERS

tree: address of a duplicated object tree.

DESCRIPTION

This function release an object tree duplicated with ObjcDup().

SEE ALSO

ObjcDup()

Programming guideline of WinDom

ObjcString()

NAME

ObjcString - get and set the label of an object.

PROTOTYPAGE

char *ObjcString(OBJECT *tree, int index, char *newstr);

PARAMETERS

tree:
address of object tree,

index:
object index,

newstr:
new label or NULL,

return:
address of the object label.

DESCRIPTION

ObjcString() provides an universal acces of the label (text) of any object. The object can be a button, a string, an icon. If the object has not text, the function does nothing. If a **NULL** value is given to *newstr* parameter, ObjcString() returns the address of the object text. The text returned can be read or modified:

```
printf( "Object i : %s\n", ObjcString( tree, i, NULL), i);
strcpy( ObjcString( tree, i, NULL), "New text");
```

It is possible to define a new buffer for the object:

```
char txt[120] = "New text";
ObjcString( tree, i, txt);
```

ObjcString() should never be used to change the text of an menu item. For that purpose, use MenuText() instead of ObjcString().

SEE ALSO

MenuText()

Programming guideline of WinDom

ObjcStrCpy()

NAME

ObjcStrCpy - Copy the label of an object.

PROTOTYPAGE

ObjcStrCpy(OBJECT *tree, int index, char *src);

PARAMETERS

tree:
address of object tree,

index:
object index,

src:
source string to copy.

DESCRIPTION

ObjcStrCpy() is just a macro function of ObjcString(). Instead of write :

```
strcpy( ObjcString( tree, obj, NULL), "new label");
```

write :

```
ObjcStrCpy( tree, obj, "new label");
```

It is the common way to use ObjcString().

SEE ALSO

ObjcString()

Programming guideline of WinDom

Resource library

Programming guideline of WinDom

RsrcLoad()

NAME

RsrcLoad - Load a resource file in memory.

PROTOTYPAGE

int RsrcLoad(**char** *rsrcfile);

PARAMETERS

rsrcfile:
file name of GEM resource,

return:
1 if no error, 0 else.

DESCRIPTION

RsrcLoad() replaces the AES rsrc_load() function. The resource filename can be in a TOS format (e.g. with backslash characters) or MiNT format (e.g. with slash characters). RsrcLoad() uses conv_path() to make the convertion. RsrcLoad() fills in the *app*.*ntree* variable with the number of tree contained in the resource file.

SEE ALSO

RsrcFree(), conv_path()

Programming guideline of WinDom

RsrcFree()

NAME

RsrcFree - Release from memory the resource.

PROTOTYPAGE

int RsrcFree(**char** *rsrcfile);

PARAMETERS

return: 0 if no error.

DESCRIPTION

RsrcFree() releases from memory the resource loaded by RsrcLoad(). This function replaces the AES rsrc_free() function.

SEE ALSO

RsrcFree()

Programming guideline of WinDom

RsrcXtype()

NAME

RsrcXtype - Install/remove extended objects.

PROTOTYPAGE

void RsrcXtype(**int** mode, OBJECT **trindex, **int** ntree);

PARAMETERS

mode:

- RSRC_XTYPE : install new type for objects having an extended type,
- RSRC_X3D : install new type for all objects,
- RSRC_XALL : cumul previous modes,
- O : uninstall all new types.

trindex:
address of all object tree or NULL,

ntree:
number of object tree in the memory resource.

DESCRIPTION

RsrcXtype() creates the special WinDom extended objects. It can work on internal resource, loaded by RsrcLoad() or on external resource, include in the C-source during the compilation.

The *mode* parameter how new objects are installed. Mode **RSRC_XTYPE** install new type for objects having an extended type. Avalaible extended types are describe in section Extended types. There is a second mode, **RSRC_X3D**, which installs new object types with a 3D look for all objects. The goal of this mode is to allow your dialog boxes to have the same aspect with MagiC, Naes, TOS or any other GEM system. Addressed objects are buttons and boxes without extended types. And off course, these two modes can be cumulated (mode **RSRC_XALL**).

To fix the objects in the internal resource, the *trindex* parameter must set to **NULL** and the *trindex* parameter is not used. To fix an external resource, the *trindex* parameter should be filled with the address of object trees in the resource. This address is supplied with the RSH file created by your resource editor. The RSH file must be include in your source code with an #include directive. The *ntree* value is also supplied with the RSH file.

When the application finish, extended object should be freed by a call to RsrcXtype() with mode 0.

SEE ALSO

Extended types for objects

Programming guideline of WinDom

RsrcFixCicon()

NAME

RsrcFixCicon - fix the color icons

PROTOTYPAGE

void RsrcFixCicon(OBJECT *tree, **int** num_obs, **int** num_cib, **int** *palette[4], void *fix);

PARAMETERS

***tree*:**
pointer to an object tree,

***num_obs*:**
number of object in *tree* ,

***num_cib*:**
nomber of color icons *tree* ,

***palette*:**
color palette,

***fix*:**
structure containing the fixed icons.

DESCRIPTION

RsrcFixCicon() fixes the color icons to the current resolution. When a resource file is loaded by RsrcLoad(), color icon are fixed by AES. The parameter *palette* is optional and can be set to NULL. This parameter is provided by the RSH file (created by resource editor program). After the call of RsrcFixCicon(), the structure *fix* is filled with the fixed icon and will be released when the program will finish by RsrcFreeCicon().

SEE ALSO

RsrcFreeCicon()

Programming guideline of WinDom

RsrcFreeCicon()

NAME

RsrcFreeCicon - Release fixed color icons.

PROTOTYPAGE

void RsrcFreeCicon (void *fix);

PARAMETER

fix:] pointer filled in by RsrcFixCicon().

DESCRIPTION

RsrcFreeCicon() should be call to release the memory reserved by RsrcFixCicon() to fix color icons.

SEE ALSO

RsrcFixCicon()

Programming guideline of WinDom

RsrcUserDraw()

NAME

RsrcUserDraw - set a drawing function to an object.

PROTOTYPAGE

int RsrcUserDraw (**int** mode, WINDOW *win, **int** index, **void** (*draw)(WINDOW *, PARMBLK *, void *), **void** *data);

PARAMETERS

mode:
OC_FORM or OC_TOOLBAR,

win:
window descriptor,

index:
object index,

draw:
drawing function,

data:
pointer to an user data,

return:
0 if no error.

DESCRIPTION

RsrcUserDraw() attaches a drawing function to an object. AES will call the function *draw* to draw the object. A drawing function has the following interface:

```
void draw( WINDOW *win, PARMBLK *pblk, void *data);
```

win is the window descriptor of the window hosting the object, *pblk* contains informations related to the object and the USERDEF structure used (object state, clip area size, object previous and current state). *data* is a pointer to a user data specified by RsrcUserDraw(). The drawing function has the following limitation:

1. objc_draw(), objc_change() and WinDom equivalent function cannot be used,
2. do not use too many local variables : remenber this function is executed by the AES and then in supervisor mode,
3. the drawing function should never clip screen because it is already performed by

WinDom.

SEE ALSO

RsrcXtype()

Programming guideline of WinDom

RsrcXload()

NAME

RsrcXload - load multiple resource file

PROTOTYPAGE

void *RsrcXload(**char** *filename);

PARAMETERS

filename:
resource file to load.

return:
0 if no error.

DESCRIPTION

As RsrcLoad(), RsrcXload() load in memory a resource file. The difference is RsrcXload() can load severals resource file. As RsrcLoad(), RsrcXload() use the PATH variable to locate the resource file (in this case, the filename should not be a pathname). The function returns a pointer which identify the resource or a NULL value if an error occurs. The pointer must be kept in memory because it is used by RsrcGaddr() to get an object address in resource and it is used by RsrcXfree() to release resource memory. Icons colors are automatically fixed by RsrcXload() using RsrcFixCicon().

EXAMPLE

```
{
        void *rsc1, *rsc2;
        OBJECT *tree;
        /* Loads resources */
        rsc1 = RsrcXload( "myrsc1.rsc");
        rsc2 = RsrcXload( "myrsc2.rsc");
        /* Get an object */
        RsrcGaddr( rsc1, R_TREE, FORM1, &tree);
        ...


        RsrcXfree( rsc2);
        RsrcXfree( rsc1);
}
```

SEE ALSO

RsrcXfree(), RsrcGaddr(), RsrcGhdr(), RsrcLoad().

Programming guideline of WinDom

RsrcXfree()

NAME

RsrcXfree - release a resource loaded by RsrcXload()

PROTOTYPAGE

void RsrcXload(**void** *rsc);

PARAMETERS

rsc:
resource to release.

DESCRIPTION

RsrcXfree() release a resource loaded by RsrcXload(). The parameter rsc is provided by RsrcXload().

SEE ALSO

RsrcXload()

Programming guideline of WinDom

RsrcGaddr()

NAME

RsrcGaddr - get an address object in a resource.

PROTOTYPAGE

int RsrcGaddr(**void** *rsc, **int** type, **int** index, **void** *addr);

PARAMETERS

rsc:
resource targeted or NULL,

type:
type of object to retrieval,

index:
index of object to retrieval,

addr:
address of object,

return:
code error.

DESCRIPTION

RsrcGaddr() has the same action than rsrc_gaddr() of resource loaded by RsrcXload(). If parameter *rsc* is NULL, RsrcGaddr() resource targeted is which loaded by RsrcLoad(). For details, see rsrc_gaddr() manual.

SEE ALSO

RsrcXload(), rsrc_gaddr().

Programming guideline of WinDom

RsrcGhdr()

NAME

RsrcGhdr - return header of a resource.

PROTOTYPAGE

rscHDR *RsrcGaddr(**void** *rsc);

PARAMETERS

rsc:
resource targeted,

return:
header address of resource or NULL if error occurs.

DESCRIPTION

RsrcGhdr() returns the header of a resource. The header is a structure containing some information about resource :

```
typedef struct {
   long            nobs;   /* number of OBJECT items */
   long            ntree;  /* number of tree OBJECT */
   long            nted;   /* number of TEDINFO items */
   long            ncib;   /* number of CICON items */
   long            nib;
   long            nbb;
   long            nfstr;  /* number of string items */
   long            nfimg;  /* number of IMAGE items */
   OBJECT          *object; /* address of OBJECTS */
   TEDINFO         *tedinfo; /* address of TEDINFO */
   ICONBLK         *iconblk; /* address of ICONBLK */
   BITBLK          *bitblk;  /* address of BITBLK */
   CICON           *cicon;   /* address of CICON */
   CICONBLK        *ciconblk; /* address of CICONBLK */
   char            **frstr;
   BITBLK          **frimg;
   OBJECT          **trindex;  /* address of tree OBJECTS */
}  rscHDR;
```

SEE ALSO

RsrcXload()

Programming guideline of WinDom

Sliders library

Programming guideline of WinDom

SlidCreate()

NAME

SlidCreate - Initialise a slider structure.

PROTOTYPAGE

void *SlidCreate(**float** min, **float** max, **float** value, **float** line, **float** page, **int** dir, **int** upd);

PARAMETERS

min:
minimal value,

max:
maximal value,

value:
current value,

line:
small incremental step,

page:
large incremental step,

dir:
direction of slider :

SLD_HORI
horizontal slider,

SLD_VERT
vertical slider.

upd:
kind of slider update :

SLD_IMME
slider is immediatly updated,

SLD_DIFF
slider is updated after the user action.

return:
slider structure created.

DESCRIPTION

The function creates a slider structure. It is not an AES object tree in a formular but only some variables to handle a set of objects representing and acting as a slider.

The object slider should be creating in a resource editor (or in other way). Then this object slider must to attached to the slider structure created by SlidCreate(). The function SlidAttach() performs this link.

Values given to SlidCreate() are flotting numbers because you can handle a decimal variable in a slider. *Min*, *max* and *value* are respectively the minimal value, the maximal value and the current value of the internal slider variable. To read or eventually to change the internal variable use functions SlidGetValue() and SlidSetValue().

Parameters *line* and *page* addresses the user interaction in the slider object. *line* represents a small incrementation or decrementation of the slider value and *page* represents a large incrementation or decrementation (see SlidAttach()).

Parameter *upd* is usefull if you attach an update function to the slider (see SlidSetFunc()). This function is called when the slider value is changed.

SEE ALSO

SlidAttach(), SlidSetValue(), SlidGetValue(), SlidSetFunc().

Programming guideline of WinDom

SlidAttach()

NAME

SlidAttach - attach a slider to an object structure in a formular.

PROTOTYPAGE

void SlidAttach(**void** *slid, **int** mode, WINDOW *win, **int** up, **int** bg, **int** sld, **int** dn);

PARAMETERS

slid:
pointer on slider structure created by SlidCreate(),

mode:
OC_FORM or OC_TOOLBAR,

win:
window descriptor of the formular host,

up:
index of decrement widget or -1,

dn:
index of increment widget or -1,

bg:
index of pager widget or -1,

sld:
index of cursor widget or -1.

DESCRIPTION

The function attaches a slider structure to slider object in a formular. Only window and toolbar formular are handled.

Objects *up* and *dn* are generally **TOUCHEXIT SELECTABLE G_BOXCHAR** object containing an up arrow and and down arrow, if slider is vertical or left arrow and right arrow if slider is horizontal. When user clicks on these widget, slide value is increased or descreased using the small increment step defined in *line* SlidCreate() parameter. *up* and *dn* can be set to -1. In these case, the slider doesnot use these widgets.

Object *sld* should always be a children of the *bg* object. When the user click on pager object (*bg*), slider value is increased or decreased using the large increment step define in *page* SlidCreate() parameter. If *sld* and *bg* are set to -1, the slider doesnot use these widgets.

The **TOUCHEXIT** flags is required for the four objects specified in SlidAttach().

SEE ALSO

SlidCreate().

Programming guideline of WinDom

SlidSetFunc()

NAME

SlidSetFunc() - define a slider event function.

PROTOTYPAGE

void SlidSetFunc(**void** *slid, **void** (*func)(), **void** *data);

PARAMETERS

slid:
pointer to a slider structure,

func:
pointer to a slider event function,

data:
pointer to a user data.

DESCRIPTION

This function attaches a slider event function to a slider. When the slider value is modified, this function is called with the following prototype :

void doslid(WINDOW *win, **int** mode, **float** value, **void** data);

win and *mode* are respectively the window descriptor host and the formular type (OC_FORM or OC_TOOLBAR). Parameter *value* is the new value of the slider. *data* is an optional pointer to a user data specified by SlidSetFunc().

SEE ALSO

SlidCreate().

Programming guideline of WinDom

SlidSetValue()

NAME

SlidSetValue : set the internal slider value.

PROTOTYPAGE

void SlidSetValue(**void** *slid, **float** value);

PARAMETERS

slid:
pointer to a slider structure,

func:
pointer to a slider event function,

data:
pointer to a user data.

DESCRIPTION

This function sets the internal slider value. If the slider is attached to a formular, the slider event function (see SlidSetFunc()) will be invoked if needed.

SEE ALSO

SlidCreate(), SlidAttach(), SlidSetFunc(), SlidGetValue().

Programming guideline of WinDom

SlidGetValue()

NAME

SlidGetValue - returns the value of a slider.

PROTOTYPAGE

float SlidGetValue(**void** *slid);

PARAMETERS

slid:
pointer to a slider structure,

return:
the internal value of the slider.

SEE ALSO

SlidSetValue().

Programming guideline of WinDom

SlidSetSize()

NAME

SlidSetSize - set the size of a slider cursor widget.

PROTOTYPAGE

void SlidSetSize(**void** *slid, **int** size);

PARAMETERS

slid:
pointer to a slider structure,

size:
new size, a value between 0 and 1000.

DESCRIPTION

The function change the size (width for an horizontal slider and height for a vertical slider) of the slider cursor widget. A value between 0 and 1000 is requested, 1000 means the largest possible size (it is the size of the cursor parent widget). The cursor and it root element are redrawn.

Generally, a cursor have a fixed size. But in the case of a slider associated to a window display a list a element, for example, this size can be used to symbolize the number of elements displayed in the window compare to the total of elements. In this case :

```
size = MIN( (element_seen / element_total)  * 1000, 1000)
```

Programming guideline of WinDom

Utility library

Programming guideline of WinDom

CallStGuide()

NAME

CallStGuide - Interface to ST-Guide.

PROTOTYPAGE

int CallStGuide(**char** *pattern));

PARAMETERS

pattern:
string to find in ST-Guide files,

return:
0 if no error, -1 si ST-Guide not in memory.

DESCRIPTION

CallStGuide() sends a **VA_START** message to ST-Guide. ST-Guide has to be loaded in memory. The parameter *pattern* can be a the path of a file or a simple pattern searched in the ST-Guide indexes.

EXAMPLE

```
/* Display in ST-Guide a simple text file */
CallStGuide( "C:\\NEWDESK.INF");
/* Find the WINDOM.HYP file in ST-Guide paths and display it */
CallStGuide( "*:\\WINDOM.HYP");
/* Find in the WINDOM.HYP file the CallStGuide() reference and display it */
CallStGuide( "*:\\WINDOM.HYP CallStGuide()");
/* Find in the ST-Guide indexes a simple reference */
CallStGuide( "A simple reference");
```

SEE ALSO

(!url [St-Guide documentation] [ST-GUIDE.HYP]), Galloc()

Programming guideline of WinDom

ShelWrite()

NAME

ShelWrite - Launch application.

PROTOTYPAGE

int ShelWrite(**char** *prg, **char** *cmd, **void** *env, **int** av, **int** single);

PARAMETERS

prg:
file to execute,

cmd:
command line or empty string,

env:
environ string or NULL,

av:
if TRUE, send a **VA_START** message if needed,

single:
if TRUE, execute in single mode,

return:
the id process of the application launched or -1 if error.

DESCRIPTION

ShelWrite() launches application in a easy way. Applications can be GEM application or TOS application. ShelWrite() uses the file name extension to identify the type of application (TOS programs have TOS or TTP suffix and GEM programs have APP, PRG or GTP suffix).

In a multitask environnment, applications are launched in parallel. With the parameter *single*, applications are lauched in single mode. If the parameter *av* is set to TRUE, a **VA_START** message is sent to the GEM application if it is running. Otherwise, a FALSE value has the consequent to create multiple application.

The parameter *cmd* describe the command line given to the application. The format is different of Pexec() or shel_write() (ie the first character does not contain the lenght of the string).

This function was backwardly nammed 'ExecGemApp()'.

BUGS

The ARGV protocol is not handled, the desktop accessores can not be launched.

Programming guideline of WinDom

GrectCenter()

NAME

GrectCenter - center an area to screen.

PROTOTYPAGE

void GrectCenter(**int** w, **int** h, **INT16** *x, **INT16** *y)

PARAMETERS

w, h:
width abd height of the area (input),

x, y:
coordinate of the centered area (output).

DESCRIPTION

GrectCenter() computes the coordinates of an area in order to centering it at screen. This function is used by WindOpen(), window formular and classic formular functions to center the window or the formular at screen. GrectCenter() replaces the AES function form_center().

User can change the way GrectCenter() centers the areas by editing the variable windom.window.center in the WinDom configuration file. Windows and formulars can be centered at screen, centered on the mouse sprite, etc.

SEE ALSO

WindOpen(), FormCreate(), WindFormBegin(), FormBegin(), form_center(), windom.window.center.

Programming guideline of WinDom

debug()

NAME

debug - trace a WinDom client.

PROTOTYPAGE

void debug(**char** *format, ...));

PARAMETERS

See printf() parameters.

DESCRIPTION

debug() is a very primitive function devoted to debugging programs. It sends a message to a special application (DEBUG), this application displays the values requested by debug(). debug() uses the variable windom.debug in the WinDom configuration file. If this variable is not defined, debug() does nothing. For more details see the documentation of windom.debug and DEBUG (in the WinDom Developper Kit).

Programming guideline of WinDom

keybd2ascii()

NAME

keybd2ascii - get the ascii code of a keyboard event.

PROTOTYPAGE

int keybd2ascii(**int** keybd, **int** shift);

PARAMETERS

keybd:
keyboard scancode provides by evnt_keybd() or **MU_KEYBD** event,

shift:
should be set to 1 if the shift key is depressed, 0 else,

retour:
tyhe ascii code associated to the event.

DESCRIPTION

keybd2ascii() identify the real ascii code of a keyboard event even the shift, control and alternate keys are depressed. When these keys are used, the ascii code in the scancode (value returned by a keyboard event) are different. Moreover, the scancode depends on the country of the keyboard. This function gets the real ascii code of the key pressed and can be used for keyboard shortcut.

EXAMPLE

```
#include <windom.h>
#include <scancode.h> /* definition of keyboard scancodes */

void ex_keybd( WINDOW *win) {
   char key =  keybd2ascii( evnt.keybd, evnt.mkstate & (K_LSHIFT|K_RSHIFT));
   switch( key) {
   case 'w':
   case 'W':
           /* key w */
           if( evnt.mkstate & K_CTRL)
                   ; /* key Control-w */
           break;
   /* ... */
   default:
           /* Some keys have no ascii code (function key, numeric pad, ...).
            * These keys can be identified by their scancode.
            */
           switch( evnt.keybd>>8) {
           case SC_HELP:
                   /* HELP key */
                   break;
           /* ... */
           }
   }
}
```

Programming guideline of WinDom

rc_set()

void rc_set(GRECT *rect, **int** x, **int** y, **int** w, **int** h);

rc_set() initializes a GRECT structure.

Programming guideline of WinDom

rc_intersect()

int rc_intersect (GRECT *r1, GRECT *r2);

rc_intersect() computes intersection between r1 and r2. r2 is filled with the intersection and the function returns 1 if the intersection exists. This function is defined in GEMLIB or PCGMXLIB but not in WINDOM library.

Programming guideline of WinDom

rc_clip_on()

int rc_clip_on (GRECT *clip);

Set the GEM clipping on. This function should be never used inside redraw function or userdraw function. After the call, coordinates of clipped area are readable in the *clip* variable. See also rc_clip_off().

Programming guideline of WinDom

rc_clip_off()

int rc_clip_off (void);

Set the GEM clipping off. This function should be never used inside redraw function or userdraw function. See also rc_clip_on().

Programming guideline of WinDom

w_get_bkgr()

NAME

w_get_bkgr - save a screen area.

PROTOTYPAGE

void w_get_bkgr(**int** x, **int** y, **int** w, **int** h, MFDB *img);

PARAMETERS

x,y,w,h:
coordinate and size of the area to save,

img:
a valid MFDB structure will containing the screen area saved.

DESCRIPTION

It is a sub function of FormBegin(). The screen area is copied in memory. A valid structure MFDB should be given to w_get_bkgr() but the memory required to save the screen area is reserved by the function. To release the memory, w_put_bkgr() should be call.

SEE ALSO

w_put_bkgr()

Programming guideline of WinDom

w_put_bkgr()

NAME

w_put_bkgr - restore a screen area

PROTOTYPAGE

void w_put_bkgr(**int** x, **int** y, **int** w, **int** h, MFDB *img);

PARAMETERS

x,y,w,h:
coordinate and size of the area to restore,

img:
a valid MFDB structure containing the screen area saved.

DESCRIPTION

It is a sub function of FormEnd(). The screen area to restore should be previously saved by w_get_bkgr(). After the call, the memory is released, so the function can be called only one time with the same MFDB structure.

SEE ALSO

w_get_bkgr()

Programming guideline of WinDom

conv_path()

NAME

conv_path - convert a file name between TOS and MiNT formats.

PROTOTYPAGE

char *conv_path(**char** *p)

PARAMETERS

p:
buffer containing the path to convert,

retour:
address of p.

DESCRIPTION

conv_path() converts a TOS filename o pathname in MiNT (Unix) format and inversly. Absolute or relative path are converted. Concerning the TOS convertion (MiNT to TOS), the root path (/ in Unix) is converted to U:\ except disk paths (/x/) which are converted into x:\ format.

EXAMPLES

"folder\dum.cnf" is converted in "folder/dum.cnf"
"/c/multitos/mint.cnf" is converted in "c:\multitos\mint.cnf"
"/etc/passwd" is converted to U:\etc\passwd
"/u/etc/passwd" is converted in "u:\etc\passwd"
"/etc/passwd" is converted in "U:\etc\passwd"

Programming guideline of WinDom

Galloc()

NAME

Galloc - Global memory reservation.

PROTOTYPAGE

void *Galloc(size_t size);

PARAMETERS

size:
memory size required,

retour:
address of buffer or NULL if error.

DESCRIPTION

Galloc() reserves global memory, i.e. memory which can be shared with other application. It is mainfully used when a message is sent to an another application: all data shared between application should be declared in global memory (or shared memory). Only MiNT with memory protection is concerned.

RESTRICTIONS

Galloc() is not a performing memory manager as malloc(), it makes a direct call to Malloc()/Mxalloc() and has the same limitation of these functions.

EXAMPLE

```
void send_fileto_qed( char *file) {
        char *path = Galloc( 128);

        strcpy( path, file);
        ApplWrite( appl_find( "QED     ", VA_START, ADR(path));
        Mfree( path);
}
```

Programming guideline of WinDom

w_getpal()

NAME

w_getpal - save in a buffer the current screen color palette.

PROTOTYPAGE

void w_getpal(W_COLOR *palette);

PARAMETERS

palette:
buffer describing the color palette or NULL pointer.

DESCRIPTION

w_getpal() saves in a buffer the current screen color palette. The buffer is an array of W_COLOR element. W_COLOR is a 3-short integer structure describing the RGB components of a color. The size of the array is given by the *app.color* global variable.

If the parameter *palette* is **NULL**, the desktop palette (*app.palette*) is used to keep the current screen color palette.

This function is used by EvntWindom() to handle desktop and window palettes depending on GEM events.

SEE ALSO

w_setpal()

Programming guideline of WinDom

w_setpal()

NAME

w_setpal - restore a screen color palette.

PROTOTYPAGE

void w_setpal(W_COLOR *palette);

PARAMETERS

palette:
buffer describing the color palette or NULL pointer.

DESCRIPTION

w_setpal() restores the screen color palette describing by the parameter *palette* i.e. this palette is applyed to the screen display.

The buffer is an array of W_COLOR element. W_COLOR is a 3-short integer structure describing the RGB components of a color. The size of the array is given by the *app.color* global variable.

If the parameter *palette* is **NULL**, the desktop palette (*app.palette*) is used to restore the screen palette.

This function is used by EvntWindom() to handle desktop and window palettes depending on GEM events.

SEE ALSO

w_getpal()

Programming guideline of WinDom

Window library

Programming guideline of WinDom

WindCreate()

NAME

WindCreate - Create a window descriptor.

PROTOTYPAGE

WINDOW *WindCreate(**int** attrib, **int** x, **int** y, **int** w, **int** h);

PARAMETERS

attrib:
bit field of window widget (CLOSER, ...),

x,y,w,h:
maximal size of the window.

return:
a pointer to a descriptor of the window created or NULL if an error occurs.

DESCRIPTION

WindCreate() is the equivalent of the AES function wind_create(). It creates a window. However, the window is not opened at screen yet. This action is performed by WindOpen().

The first parameter *attrib* defines the widget of the window. Notice that the **SMALLER** widget (devoted to iconify the window) is available with all TOS version. When this widget is not available, the window can be iconified by shift-clicking the closer widget. The **WM_BOTTOMED** is supported by WinDom for any TOS version. With TOS unsorpting this feature, a window can be 'bottomed', i.e. send to the background, by shift-clicking the mover widget.

WindCreate() remplaces completly wind_create() that means wind_create() should never be used except some very special cases. WindCreate() performes the following actions:

1. a WINDOW structure is created and inserted in the internal WinDom list (see AddWindows()),
2. the GEM window is created,
3. standard events function are attribuated to the window (via the EvntAttach() function).

The last point is very important: when a window is created by WindCreate(), this window is ready to live in the WinDom environnement, the standard functions (see next section), handle most of GEM events. Of course, these standard function are very common and basic and the developper should need to change some of these functons (for that purpose, see the Event library).

STANDARD EVENT FUNCTIONS

We list the standard functions, linked by default when a window is created with the WindCreate() function. These functions use the WindSet() function to manipulate the window.

Standard event function defined by WindCreate()

Name	Event	Description
WindClear	WM_REDRAW	Draw the window workspace background
std_cls	WM_CLOSED	send a WM_DESTROY message
std_dstry	WM_DESTROY	close an delete the window. Send an AP_TERM
		message if no more windows are opened and if
		a desktop menu is not defined.
std_tpd	WM_TOPPED	Put in foreground the window.
std_mvd	WM_MOVED	Place in right coordinate the window. Set
		win->fullsize to 0.
std_szd	WM_SIZED	Adjust the window size and update the sliders.
std_fld	WM_FULLED	Set the window in full screen or adjust the
		window to the previous position according the
		win->fullsize value.
std_icn	WM_ICONIFY	Iconify the window.
std_unicn	WM_UNICONIFY	Uniconify the window or open windows closed if
		the icon window result of a WM_ALLICONIFY event.
std_allicn	WM_ALLICONIFY	Close all opened windows and iconify the window.
std_hsld	WM_HSLID	Update the horizontal slider to new position
		(with WindSlider()). Send a WM_REDRAW evnt.
std_vsld	WM_VSLID	Update the verticatal slider to new position
		(with WindSlider()). Send a WM_REDRAW evnt.
std_arrwd	WM_ARROWED	This function used the sub function listed
		below depending the sub-mode of the message.
		These functions update the sliders, send a
		WM_REDRAW message and perform blitcopy of
		window workspace areas.
std_dnpgd	WA_DNPAGE	sub function of std_arrwd.
std_uppgd	WA_UPPAGE	idem
std_lfpgd	WA_LFPAGE	idem
std_rtpgd	WA_RTPAGE	idem
std_dnlnd	WA_DNLINE	idem
std_uplnd	WA_UPLINE	idem
std_lflnd	WA_LFLINE	idem
std_rtlnd	WA_RTLINE	idem

SEE ALSO

WindOpen(), WindClose(), WindDelete(), WindClear(), WindSet(), WindGet(), WindSlider(), GrectCenter(), AddWindow(), windom.window.center,

windom.window.effect, windom.iconify.geometry.

Programming guideline of WinDom

WindOpen()

NAME

WindOpen - Open a window.

PROTOTYPAGE

int WindOpen(WINDOW *win, **int** x, **int** y , **int** w, **int** h);

PARAMETERS

win:
address of window descriptor,

x,y,w,h:
position and size of the window.

return:
a non nul value if error.

DESCRIPTION

WindOpen() opens a window on screen and replaces the AES open_wind() function. In WinDom environnement, the open_wind() should never be used (except some very special case).

An opened window has its status **WS_OPEN** bit set to 1 (i.e. the *status* field of the WINDOW structure). If the status **WS_GROW** bit is set to 1, a graphic effect should be used (via graf_growbox() function).

It is possible to give to the *x* and *y* parameter a -1 value. In this case, the window will be horizontally centered (x=-1) or vertically centered (y=-1) or both. WindOpe() calls GrectCenter() to performe that.

A window can have minimum and maximal size : just sets the *min_w, min_h, max_w, max_h* fields of the window descriptor.

IMPORTANT

If a window has to be opened with open_wind() (for a special raison), the window should be registered in the internal WinDow list of opened windows with the add_windowlist() function. It is important for WinDom to have a correct list because the **WM_BOTTOMED** message is simuled with old TOS version.

SEE ALSO

WindCreate(), WindClose(), WindDelete(), WindClear(), WindSet(), WindGet(), GrectCenter(). add_windowlist().

Programming guideline of WinDom

WindClose()

NAME

WindClose - close a window.

PROTOTYPAGE

int WindClose(WINDOW *win);

PARAMETERS

***win*:**
address of window descriptor,

***return*:**
a non null value if error.

DESCRIPTION

WindClose() closes a window on the screen and replaces the AES wind_close function. As WindOpen() a graphic effect is used if the window status **WS_GROW** bit is set to 1.

For similar raisons explained in the WindOpen() manual, the wind_close() should never be used except in some special case. It is important to remove the window from the internal WinDom opened windows list with the remove_windowlist() function.

SEE ALSO

WindOpen(), WindCreate(), WindDelete(), WindClear() WindSet(), WindGet(), remove_windowlist().

Programming guideline of WinDom

WindDelete()

PROTOTYPAGE

int WindDelete(WINDOW *win);

PARAMETERS

***win*:**
window descriptor,

***return*:**
0 if no error occurs.

DESCRIPTION

WindDelete() remplaces wind_delete(). The window descriptor is removed from the list of windows and delete.

SEE ALSO

WindOpen(), WindClose(), WindCreate(), WindClear(), WindSet(), WindGet(), RemoveWindow().

Programming guideline of WinDom

WindSet()

NAME

WindSet - set the window parameters.

PROTOTYPAGE

/ Prototype for 16 bits compilers */*
void WindSet(WINDOW *win, **int** mode, ...);
/ Prototype for 32 bits compilers */*
void WindSet(WINDOW *win, **int** mode, **int** ap1, **int** ap2, **int** ap3, **int** ap4);

PARAMETERS

***win*:**
address of window descriptor,

***mode*:**
see **WINDSET MODE** section,

...:
depend on *mode* value.

DESCRIPTION

This function is very important. It replaces completly the wind_set() function that should be never used.

There are two prototypes of this function : one addressing 16 bits compilers and one addressing 32 bits compilers. For the second case, we use the classic four integers parameters to prevent errors from passing pointer parameters. Indeed, with 32 bits mode, integer and pointer have same size. For that reason, pointer parameters must be encapsuled with ADR() macro function. The fixing prototype of WindSet() cause warning or error if this rule is not respected. You can use also WindSetStr() or WindSetPtr() macros function with modes attenting for one or two pointer parameters.

For 16 bits compilers (Pure C and Sozobon) there is no change except this rule should respected if you want to hack portable source.

WINDSET MODE

Each mode of WindSet() are listed in the next table. The difference with wind_set() are underlined.

WindSet() modes

Mode	*Description*	*Comments*
WF_NAME	Set the window title. The	wind_set() uses the
	title is only used for	same title for iconified
	uniconified window.	or non iconified window.
	WindSet(win, WF_NAME, title);	
WF_ICONTITLE	Set the icon window title.	Specific WinDom feature.
	WindSet(win, WF_ICONTITLE,	
	title);	
WF_ICONDRAW	Set the icon drawing function	Specific WinDom feature.
	WindSet(win, WF_ICONTITLE,	A NULL value removes the
	draw);	drawing function.
WF_INFO	Set the informative bar text.	
	WindSet(win, WF_INFO, info);	
WF_WORKXYWH	Set the window workspace	WindSet() takes in
	coordinate	consideration menu and
	WindSet(win, WF_WORKXYWH,	toolbar.
	x, y, w, h);	
WF_TOP	Put the window in foreground.	The internal WinDow cycle
	WindSet(win, WF_TOP);	window is updated (for
	or any window (top = handle)	WM_BOTTOMED message
	WindSet(NULL, WF_TOP, top);	emulation).
WF_BEVENT	Set the window event behaviour	B_MODAL is a WinDom
	WindSet(win, WF_BEVENT, val);	feature. This mode is
	val is a bit field :	supported by all TOS.
	B_UNTOPPABLE: a mouse button	
	event on a window workspace	
	create a MU_BUTTON event	
	instead of a WF_TOP event.	
	B_MODAL: a window modal is the	
	only window active.	
WF_BOTTOM	Set a window in the background	This mode is supported
	WindSet(win, WF_BOTTOM);	by all TOS.
WF_ICONIFY	Iconify a window at specified	This mode is supported
	coordinate.	by all TOS.
	WindSet(win, WF_ICONIFY,	
	x, y, w, h);	
WF_UNICONIFY	Iconify a window at specified	This mode is supported

	coordinate.	by all TOS.
	WindSet(win, WF_UNICONIFY,	
	x, y, w, h);	
WF_UNICONIFXYWH	Set the coordinate of an	This mode is supported
	iconified window that will be	by all TOS.
	for the next WF_UNICONIFY	
	WindSet(win, WF_UNICONIFYXYWH,	
	x, y, w, h);	
WF_TOOLBAR	Put a toolbar in a window and	This mode is supported
	set eventually the event	by all TOS. However WinDom
	toolbar function.	has its own toolbar
	WindSet(win, WF_TOOLBAR,	management even when the
	tree, func);	system supports the
		window toolbar.
WF_MENU	Put a menu in a window and set	This mode is a specific
	eventually the event menu	WinDom feature supported
	function.	by all TOS.
	WindSet(win, WF_TOOLBAR,	
	tree, func);	
WF_HILIGHT	When a window menu is visited	This mode is a specific
	the **hilight** function is called	WinDom feature supported
	function. It is typically used	by all TOS.
	to display an online help on	
	menun function for example.	
	WindSet(win, WF_HILIGHT,	
	hilight, FUNC);	
	hilight have the proto :	
	void (*hilight)(WINDOW *win,	
	int title, int item);	
	title is the menu title visited	
	item is the menu item visited	
WF_HSLIDE	These mode are obsolets, see	
	the WindSlider() function	
WF_VSLIDE		
WF_HSLSIZE		
WF_VSLSIZE		

other mode	WindSet(win, mode, p1, p2, p3 p4);	Direct call to wind_set()

Notes about toolbar and menu

EvntWindom() draws automatically menu and the toolbar inside a window. The object tree given to WindSet() to define a menu or a toolbar is duplicated in memory using the ObjcDup() function in order to handle a same object tree in several windows. Memory is free up by WindDelete().

SEE ALSO

WindSetStr(), WindSetPtr(), WindGet(), WindSlider(), Frames

Programming guideline of WinDom

WindSetStr()

PROTOTYPAGE

int WindSetStr(WINDOW *win, **int** mode, **char** *str);

PARAMETERS

***win*:**
window descriptor,

***mode*:**
WindSet() mode,

***str*:**
string parameter.

DESCRIPTION

WindSetStr() is used intead of WindSet() with modes passing one pointer parameter : WF_TITLE, WF_ICONTITLE, WF_ICONDRAW, WF_INFO, WF_HILIGHT.

SEE ALSO

WindSet(), WindSetPtr().

Programming guideline of WinDom

WindSetPtr()

PROTOTYPAGE

int WindSetPtr(WINDOW *win, **int** mode, **void** *p1, **void** *p2);

PARAMETERS

***win*:**
window descriptor,

***mode*:**
WindSet() mode,

***p1, p2*:**
two pointer parameters.

DESCRIPTION

WindSetStr() is used intead of WindSet() with modes passing two pointer parameters : WF_MENU, WF_TOOLBAR.

SEE ALSO

WindSet(), WindSetStr().

Programming guideline of WinDom

WindGet()

NAME

WindGet() - informations about a window.

PROTOTYPAGE

void WindGet(WINDOW *win, **int** mode, ...);

PARAMETERS

win:
window descriptor,

mode:
type d'information,

...:
varie selon la valeur de *mode*.

DESCRIPTION

WindGet() replaces the AES wind_get() function. In order to exploit the special WinDom features, WindGet() should always be used instead of wind_get().

MODE

The next table lists the WindGet() modes.

WindGet() modes (under construction)

mode	Description	Comments
WF_FTOOLBAR	This mode is a WF_FIRSTXYWH mode dedicated to the toolbar redraw. WindSet(win, WF_FTOOLBAR, , &x, &y, &w, &h	With WinDom, this mode is useless. However, it can be used for higher compatibility with AES 4.
WF_NTOOLBAR	This mode is a WF_NEXTXYWH mode dedicated to the toolbar redraw. WindSet(win, WF_FTOOLBAR, , &x, &y, &w, &h	See previous remark.

SEE ALSO

WindSet(), Frame library

Programming guideline of WinDom

WindSlider()

PROTOTYPAGE

void WindSlider(WINDOW *win, **int** slider);

PARAMETERS

***win*:**
window descriptor,

***slider*:**
a bit field (actions to perform):

HSLIDER:
update horizontal slider,

VSLIDER:
update vertical slider.

DESCRIPTION

WindSlider() updates the size and position of horizontal and vertical sliders using the values of the fields *xpos, ypos, xpos_max, ypos_max, h_u* and *w_u* of the window descriptor.

SLIDERS VARIABLES

ypos
Sets the vertical slider position. It is a positive value in 0 and ypos_max: 0 <= ypos < ypos_max,

h_u
sets the vertical offset (in pixel) when a scroll event occurs,

xpos
as ypos, xpos is devoted to the horizontal slider position,

w_u
as h_u, horizontal offset.

Programming guideline of WinDom

WindCalc()

NAME

WindCalc - window coordinates computation.

PROTOTYPAGE

int WindCalc(**int** type, WINDOW *win,
int x_in, **int** y_in, **int** w_in, **int** h_in,
INT16 *xout, INT16 *yout, INT16 *wout, INT16 *hout);

PARAMETERS

***type*:**

WC_BORDER (0):
convert work area coodinate in window real coordinates,

WC_WORK (1):
inverse operation of WC_BORDER.

***win*:**
window descriptor,

***x_in, y_in, w_in, h_in*:**
input coordinates,

***xout, yout, wout, hout*:**
output coordinates,

***return*:**
a null value if error.

DESCRIPTION

This function is the wind_calc() WinDom equivalent. It takes in account the optional menu or toolbar of the window.

SEE ALSO

wind_calc().

Programming guideline of WinDom

WindHandle()

NAME

WindHandle - find a window descriptor by its AES handle.

PROTOTYPAGE

WINDOW *WindHandle(**int** handle);

PARAMETERS

***handle*:**
GEM window handle,

***return*:**
pointer to the window descriptor matching the handle or NULL.

DESCRIPTION

This function converts a GEM window handle in a WinDom window descriptor. If the window is not found, WindHandle() returns a NULL value (the window does not exist or belongs to an another application). This function is often used to analyse a GEM message or with the AES wind_find() function.

SEE ALSO

wind_find()

Programming guideline of WinDom

WindFind()

NAME

WindFind - find a window descriptor

PROTOTYPAGE

WINDOW *WindFind(**int** mode, ...);

PARAMETERS

mode:
search mode,

...:
depends on mode value :

- WDF_NAME: find a window by name, a string parameter is attented,
- WDF_INFO: find a window by info string, a string parameter is attented,
- WDF_ID: find a window by GEM handle, an integer value is attented,
- WDF_MENU:
- WDF_TOOL:
- WDF_DATA:

return:
window descriptor found or NULL.

DESCRIPTION

SEE ALSO

WindHandle()

Programming guideline of WinDom

WindTop()

NAME

WindTop - set to foreground a window.

PROTOTYPAGE

void WindFind(WINDOW *win);

PARAMETERS

win:
targetted window descriptor.

DESCRIPTION

WindTop() set to foreground a window. If the window is iconified, it will be uniconfied. If the window is opened, it will be topped. If the window is closed, it will be opened reopened at its previous location on screen.

This function is typically used with WindTop() when you doesn't want created a window already defined (as FormCreate()).

Programming guideline of WinDom

WindAttach()

NAME

WindAttach - transform an alien window in a WinDom window.

PROTOTYPAGE

WINDOW *WindAttach(**int** handle);

PARAMETERS

handle:
window handle,

return:
new window descriptor created.

DESCRIPTION

WindAttach() allows you to integrate alien windows in the WinDom environnement. The alien window should be created and opened before call of WindAttach(). This function is mainly used to insert window created by an another application inside your application. For example, FselInput() uses it to integrate the MagiC file selector as a window in the application.

SEE ALSO

AddWindows(), RemoveWindows(), remove_windowlist(), add_windowlist().

Programming guideline of WinDom

WindClear()

NAME

WindClear() - draws the window background.

PROTOTYPAGE

void WindClear(WINDOW *win);

PARAMETERS

win:
window descriptor.

DESCRIPTION

This function draws the window background i.e. a bar (typically with a white color) in the work area of a window. This function should be alway used by developer as the first call of a custom redraw event function because the user can parametrize the style and the color vi the configuration file. WindClear() is used as default redraw event function by WindCreate().

SEE ALSO

windom.window.bg

Programming guideline of WinDom

add_windowlist()

NAME

add_windowlist() - add a window in the cycle window list.

PROTOTYPAGE

void add_windowlist(**int** handle);

PARAMETERS

handle: window GEM handle to include.

DESCRIPTION

This function is a sub function of WindOpen() and it is devoted to the **WM_BOTTOM** message emulation. It should be used if you does not use WindOpen() to open your windows. Use this function at your own risk.

SEE ALSO

remove_windowlist(), WindAttach().

Programming guideline of WinDom

remove_windowlist()

NAME

remove_windowlist() - remove a window from the cycle window list.

PROTOTYPAGE

void remove_windowlist(**int** handle);

PARAMETERS

handle: window GEM handle to remove.

DESCRIPTION

This function is a sub function of WindClise() and it is devoted to the **WM_BOTTOM** message emulation. It should be used if you does not use WindClose() to close your windows. Use this function at your own risk.

SEE ALSO

add_windowlist(), WindAttach().

Programming guideline of WinDom

AddWindow()

NAME

AddWindow - Add a window descriptor in the WinDom windows list.

PROTOTYPAGE

void AddWindow(WINDOW *win)

PARAMETRE

win: window descriptor.

DESCRIPTION

This function is a sub function of WindCreate(). It should be used if you create your own window without use WindCreate(). Use this function at your own risk! Notice that the WindAttach() function allows you to include an alien window in the WinDom window environment.

SEE ALSO

WindCreate(), WindAttach(), RemoveWindow().

Programming guideline of WinDom

RemoveWindow()

NAME

RemoveWindow - remove a window descriptor in the WinDom windows list.

PROTOTYPAGE

void RemoveWindow(WINDOW *win)

PARAMETER

win: window descriptor to remove.

DESCRIPTION

This function is a sub function of WindDelete(). It should be used if you delete a window which didn't create with WindCreate() (in this case, the WindDelete() function is forbiden). Use this function at your own risk! Notice that the WindAttach() function allows you to add ans remove an alien window in the WinDom environment.

SEE ALSO

WindDelete(), WindAttach(), AddWindow().

Programming guideline of WinDom

Macros, constantes, structures, ...

Programming guideline of WinDom

Macro functions

Programming guideline of WinDom

ADR()

ADR(ptr) transforms a pointer parameter *ptr* into two integer parameters (see ApplWrite() and WindSet() manuals).

Programming guideline of WinDom

MIN()

MIN(a,b) returns the minimum of *a* and *b*,

Programming guideline of WinDom

MAX()

MAX(a,b) returns the maximum of *a* and *b*.

Programming guideline of WinDom

FORM()

FORM(win) returns the object tree address of the *win* window formular.

Programming guideline of WinDom

TOOL()

TOOL(win) returns the object tree address of the toolbar of the *win* window.

Programming guideline of WinDom

IS_IN()

IS_IN(mx,my,x,y,w,h) returns TRUE if the point (mx,my) belongs to the square (x,y,w,h).

Programming guideline of WinDom

SET_BIT()

NAME

SET_BIT - bits field handling

PROTOTYPAGE

SET_BIT(field, bit, value) /* macro function */

PARAMETERS

field:
bit field variable,

bit:
bit to set,

value:
if 1 the bit is set to 1, if 0 the bit is set to 0.

DESCRIPTION

SET_BIT() is an usefull macro function used to set and used specific bit in a variable; It is very appropriate in formular handling.

As SET_BIT() is a macro, the variable *field* does not need to be a pointer.

EXAMPLES

```
// the object SAVE state should be SELECTED is option has a SAVE bit set to 1
SET_BIT( tree[SAVE].ob_state, SELECTED, option & SAVE);

// inverse operation
SET_BIT( option, SAVE, tree[SAVE].ob_state & SELECTED);
```

Programming guideline of WinDom

Global variables and data structures

Programming guideline of WinDom

WINDOW

A window in a WinDom program is identified by a window descriptor. It is a pointer on a WINDOW structure :

```
typedef struct _window {
  int           handle;         /* AES Handle of the window */
  int           attrib;         /* window widgets */
  int           status;         /* WinDom status (see Window status flags) */
  W_GRAFPORT    graf;           /* VDI workstation opened for the window */
  W_MENU        menu;           /* menu ressources */
  W_FORM        tool;           /* toolbar ressources */
  W_ICON        icon;           /* iconified window related */
  GRECT         createsize;     /* Original window size */
  char*         name;           /* window  name */
  char*         info;           /* information bar */
  INT16         w_max, h_max;   /* maximal window size */
  INT16         w_min, h_min;   /* minimal window size */
  long          xpos, ypos;     /* relative data position in the window */
  long          xpos_max, ypos_max; /* Maximal values of previous variables */
  INT16         w_u, h_u;       /* vertical and horizontal scroll offset */
  struct _window *next;         /* next window */
  int type;                     /* user window type */
  void *data;                   /* window data - reserved */
  void *binding, last;          /* window events - reserved */
} WINDOW, *WINDOWPTR;
```

Programming guideline of WinDom

WINDOW wglb

To handle the list of windows, WinDom use a global variable :

```
typedef struct {
  WINDOW *first;     /* First window */
  WINDOW *front;     /* Topped window */
  WINDOW *appfront; /* Relative application topped window */
} WINvar;
extern WINvar wglb;
```

Each field of wglb can be NULL. The front window is the window in the foreground, it could be NULL if the topped window don't belong to our application. The appfront window is the topped window in our application but a window of an another application may be in the foreground.

Programming guideline of WinDom

W_FORM

```
typedef struct {
  OBJECT *root,     /* Address of object tree (can be duplicated) */
         *real;     /* Address of the real object tree (used by BubbleEvnt()) */
  int    *save;     /* Copy of objects' state (used by FormSave()) */
  int    edit,      /* index of the current editable field */
         nb_ob;     /* number of objects of the formular */
  INT16  cursor;    /* cursor position in the current editable field */
} W_FORM;
```

This structure is used by the window formulars and window toolbar. The *win->data* field of the window formular points to this structure.

Programming guideline of WinDom

W_GRAFPORT

```
typedef struct _grafport{
  INT16    handle;   /* VDI virtual workstation handle */
  W_COLOR *palette;  /* Color palette of the workstation */
} W_GRAFPORT;
```

Remember in WinDom each window has its own VDI workstation. The desktop has its own VDI workstation too.

Programming guideline of WinDom

W_MENU

```
typedef struct {
  OBJECT *root;       /* Menu object tree */
  int     scroll;     /* Menu scroller widget relative position */
  void *bind;
  void (*hilight)( struct _window *, int, int);
} W_MENU;
```

Programming guideline of WinDom

W_ICON

```
typedef struct {
  char *name;         /* name of the window if iconifyed */
  INT16 x, y, w, h;  /* coordinate and size of the uniconified window */
  void (*draw)(struct _window *); /* The drawing icon function */
} W_ICON;
```

Programming guideline of WinDom

W_COLOR

```
typedef int    W_COLOR[3];
```

W_COLOR is a type used to handle the palet colors of desktop and windows in non True color display modes.

Programming guideline of WinDom

INT16

INT16 is a 16-bit integer. It is defined by MGEMLIB to handle gcc 32 and 16 bits modes with GEM libraries.

Programming guideline of WinDom

struct w_version WinDom

This variable describes the current version of WinDom library.

```
extern
struct w_version {
    short patchlevel;   /* Major number version : 0x120 stands for 1.20 */
    short release;              /* Minor number version (begining at 1) */
    char *date;                 /* Date of compilation */
    char *time;                 /* Time of compilation */
    char *cc_name;              /* Name of compiler used can be :
                                                        "Pure C"
                                                        "Gnu C"
                                                        "Sozobon X"
    short cc_version;   /* Number version of compiler used */
} WinDom;
```

Fields *patchlevel* and *release* are new from WinDom version 1.20 ?

Programming guideline of WinDom

APPvar app

The global variables used by WinDom are grouped in a structure.

```
typedef struct _APPvar {

        /* Private structure which containing configuration.
     * Configuration is now performed via function ApplGet/ApplSet */

        void *config;

    /* system information variables */

    int     id;             /* AES application handle */
    INT16   handle;         /* VDI workstation desktop handle */
    INT16   aeshdl;         /* VDI workstation handle used by AES */
    INT16   x,y,w,h;        /* Size and coordinate of the desktop */
    int     color;          /* number of available colors */
    OBJECT  *menu;          /* address of the desktop menu */
    W_COLOR *palette;       /* Application palette color */
    INT16   work_in[10];    /* VDI default workstation initializer */
    INT16   work_out[57];   /* VDI workstation opening results */
    int     aes4;           /* Special AES4 features (see AES4_ constants) */
    int     gdos;           /* Gdos indicator and number of available fonts */
    int     avid;           /* AES handle of the AV-server */
    int     ntree;          /* Number of object trees in the loaded ressource */
    char    *pipe;                      /* a 256-buffer in global memory ready to use */

        /* Private structures */

    void    *binding;
    void    *hilight;
    void    *mnbind;

} APPvar;

extern APPvar app;
```

Programming guideline of WinDom

EVNTvar evnt

```
typedef struct {
  long  timer;                              /* MU_TIMER parameter */
  int   bclick, bmask, bstate;              /* MU_BUTTON parameters */
  int   m1_flag, m1_x, m1_y, m1_w, m1_h;    /* MU_M1 parameters */
  int   m2_flag, m2_x, m2_y, m2_w, m2_h;    /* MU_M1 parameters */
  INT16 buff[8];                            /* Result of MU_MESAG event */
  INT16 mx, my, mbut, mkstate;              /* Results of  MU_BUTTON */
  INT16 keybd, nb_click;                    /* and MU_KEYBD events */
} EVNTvar;

extern EVNTvar evnt;
```

This structure is used by EvntWindom() to call evnt_multi() and to store the events informations.

Programming guideline of WinDom

GRECT clip

```
GRECT clip;
```

This variable contains the coordinate and size of the current clipped zone during a WM_REDRAW event update. It is used by redraw functions to optimize the redram operations.

Programming guideline of WinDom

Constants of some bitfield variables

Programming guideline of WinDom

The app->aes4 variable

This variable provides some informations about your operating system required by WinDom. This informations are given by the AES appl_getinfo() function.

AES4 flags

Name	*Value*	*Signification*
AES4_BOTTOM	0x0001	AES handles the WM_BOTTOM message
AES4_ICONIFY	0x0002	AES handles the WM_(UN)ICONIFY messages
AES4_ICONIFYXYWH	0x0004	AES handles the WM_ICONIFYXYWH message
AES4_SMALLER	0x0008	the widget SMALLER (iconfier) is available
AES4_BOTTOMER	0x0010	the widget BOTTOMER (backdropper) is available
AES4_APPSEARCH	0x0020	the AES appl_search() function is available

Programming guideline of WinDom

WINDOW status variable

Window status flags

Name	*Value*	*Signification*
WS_OPEN	0x0001	The window is opened
WS_ICONIFY	0x0002	The window is iconified
WS_MENU	0x0004	The window owns a menu
WS_TOOLBAR	0x0008	The windpw owns a toolbar
WS_GROW	0x0010	The growbox effects are enabled
WS_UNTOPPABLE	0x0020	The window is untoppable
WS_FORM	0x0040	The window is an object formular
WS_FORMDUP	0x0080	The window is an dupicated object formular
WS_MODAL	0x0100	The window is modal
WS_FRAME_ROOT	0x0200	The window containts framed windows
WS_FRAME	0x0400	The window is a framed window
WS_ALLICNF	0x0800	The window is iconified or closed (*)
WS_FULLSIZE	0x1000	The window has full screen size

(*) This flag is used to handle the WM_ALLICONIFY message.

Programming guideline of WinDom

code error

Many WinDom functions return a negative code error. In this version, WinDom tries to standardize this errors to TOS code errors. However, some functions return yet non standardized errors. It should be fixed in the future... The WinDom package supplies an header file (toserror.h) which describe all TOS errors.

E_OK
no error,

EBADRQ
bad request (one or several parameters are not valid),

ERANGE
range error,

ENSMEM
insufficient memory,

EFILNF
file not found.

Programming guideline of WinDom

GEM extensions

Modern GEM library such as (M)GemLIB and PCGMXLIB provide new GEM function binding. We describe some of them in this section.

Extended GEM function manuals

Programming guideline of WinDom

Extended GEM function manuals

Programming guideline of WinDom

appl_getinfo()

NOM

appl_getinfo() - gives information about AES.

PROTOTYPAGE

int appl_getinfo(**int** mode, **int** *out1, **int** *out2, **int** *out3, **int** *out4);

AVAILABILITY

If call appl_find("?AGI") returns -1. If call has_applgetinfo() returns 1; If flag AES$4 app.aes4 is set to 1.

DESCRIPTION

WinDom uses this function to know the specific features of the AES and eventually use them. These specific features are held in the *app.aes4* variable.

Remark: To know if this function is available on your system, use the function has_app_getinfo().

SEE ALSO

has_appl_getinfo()

Programming guideline of WinDom

appl_search()

NOM

appl_search() - identification of GEM processes.

PROTOTYPAGE

int appl_search(**int** mode, **char** *fname, **int** type, **int** ap_id);

PARAMETERS

mode:
0 (first process), 1 (next process),

fname:
process name (a 8-character string eventually filled with space characters)

type:
process type (bit fiold):

- 0x01: system process,
- 0x02: application,
- 0x04: desktop accessory,
- 0x08: desktop.

ap_id:
process GEM id,

return:
0 if no more process to list.

DESCRIPTION

This function is avalaible in PCGEMLIB.LIB from version 1.1 of Pure C. The **AES4_APPSEARCH** bit of *app.aes4* variable is set to 1 if appl_search() is avalaible. The function is usually used to list the GEM processes.

SEE ALSO

ApplName(), appl_getinfo()

Programming guideline of WinDom

appl_control()

NAME

appl_control() - applicaton control (NAES function)

PROTOTYPAGE

int appl_control(**int** ap_cid, **int** ap_cwhat, **void** *ap_cout);

PARAMETERS

ap_cid, ap_cwhat:
see ApplControl(),

ap_cout :
unused.

retour:
0 if error, >0 else.

DESCRIPTION

This function is only avalaible with Naes. Prefer, if possible, the universal ApplControl() function.

SEE ALSO

ApplControl(), vq_naes().

Programming guideline of WinDom

objc_sysvar()

NOM

objc_sysvar() - identification des processus GEM.

PROTOTYPAGE

int objc_sysvar(**int** mode, **int** which, **int** in1, **int** in2, **int** *out1, **int** *out2);

PARAMETERS

mode:

fname:

ap_id:

retour:

DESCRIPTION

under construction ...

SEE ALSO

Programming guideline of WinDom

fslx_do()

NOM

fslx_do - call the extended file selector.

PROTOTYPAGE

```
void * fslx_do( char *title, char *path, WORD pathlen, char *fname,
                        int fnamelen, char *patterns, XFSL_FILTER *filter,
                        char *paths, int *sort_mode, int flags,
                        int *button, int *nfiles, char **pattern );
```

NOTES

This function is available when the bit **AES4_FSLX** of *app.aes4* is set to one. If possible, FselInput() uses this function to call the file selector.

SEE ALSO

FselInput(), fslx_open(), fslx_close(), fslx_evnt()

Programming guideline of WinDom

fslx_open()

NOM

fslx_open - open the file selector inside a window.

PROTOTYPAGE

```
void *fslx_open( char *title, int x, int y, int *handle,
                                char *path, int pathlen,
                                char *fname, int fnamelen,
                                char *patterns, XFSL_FILTER *filter,
                                char *paths, int sort_mode, int flags);
```

NOTES

This function is available when the bit **AES4_FSLX** of *app.aes4* is set to 1. If possible, FselInput() uses this function to call the file selector.

SEE ALSO

FselInput(), fslx_do(), fslx_close(), fslx_evnt()

Programming guideline of WinDom

fslx_evnt()

NOM

fslx_evnt - handle the GEM events of the windowing file selector.

PROTOTYPAGE

```
int fslx_evnt( void *fsd, EVNT *events,
               char *path, char *fname,
                      int *button, int *nfiles,
               int *sort_mode, char **pattern );
```

NOTES

This function is available when the bit **AES4_FSLX** of *app.aes4* is set to 1. If possible, FselInput() uses this function to call the file selector.

SEE ALSO

FselInput(), fslx_do(), fslx_close(), fslx_open()

Programming guideline of WinDom

fslx_close()

NOM

fslx_close - close the windowing file selector.

PROTOTYPAGE

int fslx_close(void *fsd);

NOTES

This function is available when the bit **AES4_FSLX** of *app.aes4* is set to 1. If possible, FselInput() uses this function to call the file selector.

SEE ALSO

FselInput(), fslx_do(), fslx_evnt(), fslx_open()

Programming guideline of WinDom

Convert your old WinDom applications

Current version of WinDom is 1.20 (October 2002)

Programming guideline of WinDom

From WinDom version 1.10 (September 2001)

Frame structures

As frame structures have been removed from public acces, we use FrameGet() to acces information.

Remplace :

```
line = win->frame.line;
col = win->frame.col;
```

by

```
FrameGet( win, WF_CELL, &line, &col);
```

Programming guideline of WinDom

From WinDom version 1.00 (November 2000)

Some macros and functions had changed

- remplace 'min()' by 'MIN()'
- remplace 'max()' by 'MAX()'
- remplace 'is_in()' by 'IS_IN()'
- remplace 'STR2INT()' by 'ADR()'
- remplace 'ExecGemApp()' by 'ShelWrite()'
- remplace 'rect_set()' by 'rc_set()'
- remplace 'set_clip()' by 'rc_clip_on()'
- remplace 'clip_off()' by 'rc_clip_off()'

Pointers parameters in some function with Gcc 32

- emcapsule pointer arguments with ADR() macro in ApplWrite() function. For example,

```
strcpy( app.file, file_to_open);
ApplWrite( apid, VA_START, app.pipe);
```

  is remplaced by :

```
strcpy( app.file, file_to_open);
ApplWrite( apid, VA_START, ADR(app.pipe));
```

  The variable *app.file* is a 256-character buffer reserved in global memory by ApplInit() and devoted to GEM communication with other application (very important with MiNT memory protection mode).

- The previous remark addresses the WindSet() function. Use the macro function ADR() or WindSetPtr() and WindGetPtr(). functions.

New macro SET_BIT

The function set_bit() is remplaced by the macro function SET_BIT(). The main advantage is the macro uses untyped variable. The first parameter of set_bit() was a pointer but in SET_BIT() it is not a pointer. Let's see an example. Remplace :

```
int val;
set_bit( &val, 0x100, TRUE);
```

by :

```
int val;
```

```
SET_BIT( val, 0x100, TRUE);
```

List of functions with new prototype

Many functions have new prototype (INT16 type instead of int type). For Pure C users, there is no change. Functions concerned are:

AvWaitfor(), ObjcEdit(), WindGet(), ApplWrite(), WindCalc(), give_iconifyxywh(), vqt_extname(), RsrcFixCicon(), FrameCalc(), GrectCenter().

Programming guideline of WinDom

From WinDom version March 2000

Data attach

Up to WinDom version March 2000, WinDom could handle two differents data per window (using the fields *data* and *data2* of the WINDOW structure). WinDom uses now a new method to attach data to window. The number of data is illimited. The field *data* is the root item of a list of data. The field *data2* is obsolet and has been removed. Data attachment is handled using the functions DataAttach(), DataSearch() and DataDelete(). All data are identified by a magic number (as cookies). Get an example :

```
/* Here our data */
typedef
struct _mydata {
        int i;
        char c;
        float f;
} MYDATA;

{
        WINDOW *win;
        MYDATA *data = malloc(sizeof(MYDATA));

        /* Attach a Data to a window */

        /* old way      */                          /* New way */
        win->data = data;                       DataAttach( win, 'DAT1', data);
        win->data2= data;                       DataAttach( win, 'DAT2', data);

        /* Get Data */

        /* old way */                           /* New way */
        display( win->data);            display( DataSearch( win, 'DAT1'));
        display( win->data2);           display( DataSearch( win, 'DAT2'));
}
```

The field *type*, which identify a window, is kept for backward compatiblity. It is just a user variable, not used by WinDom.

Timer parameters The variable evnt.lo_timer and evnt.hi_timer are now replaced by the variable evnt.timer:

```
        long timer;

        evnt.lo_timer = (int)timer;
        evnt.hi_timer = (int)(timer>>16);
```

is replaced by :

```
        long timer;

        evnt.lo_timer = timer;
```

Programming guideline of WinDom

From WinDom June 1999

Event handling

The main different with old version of WinDom and the new one is the Event handling. Before, each window had a set of pointer matching a specific event. For example, the pointer *win->redraw* matched the **WM_REDRAW** event. In the new version, you can associate any event to any window (and more :)). For details, reads the EvntAttach() manual.

Old way	New way
win->redraw = redraw;	EvntAttach(win, WM_REDRAW, redraw);
win->destroy = destroy;	EvntAttach(win, WM_DESTROY, destroy);
win->closed = closed;	EvntAttach(win, WM_CLOSED, closed);
win->fulled = fulled;	EvntAttach(win, WM_FULLED, fulled);
win->sized = sized;	EvntAttach(win, WM_SIZED, sized);
win->moved = moved;	EvntAttach(win, WM_MOVED, moved);
win->topped = topped;	EvntAttach(win, WM_TOPPED, topped);
win->untopped = untopped;	EvntAttach(win, WM_UNTOPPED, untopped);
win->iconified = icon;	EvntAttach(win, WM_ICONIFY, icon);
win->uniconified = unicon;	EvntAttach(win, WM_UNICONIFY, unicon);
win->alliconified = allicon;	EvntAttach(win, WM_ALLICONIFY, allicon);
win->hslided = hslided;	EvntAttach(win, WM_HSLID, hslided);
win->vslided = vslided;	EvntAttach(win, WM_VSLID, vslided);

The special case of WM_ARROWED

The WA_UPLINED, WA_DNLINED, WA_LFLINED, WA_RTLINED, WA_UPPAGED, WA_DNPAGED, WA_LFPAGED, WA_RTPAGED are sub messages of the WM_ARROWED message. In the new WinDom version, it is only possible to attach the WM_ARROWED message :

Old way:

```
win->uppaged = uppage;
win->dnpaged = dnpage;
win->uplined = upline;
win->dnlined = dnline;
win->lfpaged = lfpage;
win->rtpaged = rtpage;
win->lflined = lfline;
win->rtlined = rtline;
```

New way:

```
EvntAttach( win, WA_ARROWED, arrow);

/* where arrow() is defined by: */
```

```
void arrow( WINDOW *win) {
        switch( evnt.buff[4]) {
        case WA_UPPAGED:
                uppage( win); break;
        case WA_DNPAGED:
                dnpage( win); break;
        /* etc ... */
        }
}
```

snd_msg()

This function is obsolet, use the more flexible and generic function ApplWrite(). The calls :

```
        snd_msg( win, msg, w4, w5, w6, w7);
        snd_msg( NULL, msg, w4, w5, w6, w7);
```

are replaced by :

```
        ApplWrite( app.id, msg, win->handle, w4, w5, w6, w7);
        ApplWrite( app.id, msg, w4, w5, w6, w7);
```

With ApplWrite, send a message is really easy. Example :

```
        ApplWrite( appl_find( "QED     ", VA_START, "C:\\NEWDESK.INF");
```

win->fullsize

This field in the WINDOW structure has been removed. It is remplaced by the bit **WS_FULLSIZE** in the status field.

The sequence :

```
        if( win->fullsize)
                printf( "full screen window);
```

is replaced by:

```
        if( win->status & WS_FULLSIZE)
                printf( "full screen window);
```

Programming guideline of WinDom

Frequently Asked Questions

---

1. Keyboard events with keys 1, 2, 3, 4, 5, 6 from the numerical pad are not detected by WinDom. Is a bug ?

   No, it is not a bug. It is probably due the application uses a menu created by Interface. When Interface creates a new, it gives to the accessory items in the menu the following names : Accessory 1, Accessory 2, ... Unfortunately, EvntWindom() - which handles automatically the menu shortcuts - interpretes the words 1, 2, 3 of accessory items as shortcut. It is why there are not interpreted as keybord event but as menu event. The solution is to give an another name to this accessory items. For example, Accessory_1. (Thanx to Zerkman for the solution).

2. How control the redraw message, i.e. how disable the WinDom feature which clip and call the redraw function on each element of the AES rectangle list ?

   You have to bind the WM_PREREDRAW event instead of the WM_REDRAW event. You should remove the standard redraw function binded to WM_REDRAW (with EventDelete()).

Programming guideline of WinDom

Comparison of AES functions and WinDom functions

Some GEM functions shouldn't be used in the WinDom environment. The folowing table lists compatibilities beetwen AES functions and WinDom functions.

Compatibilities of AES functions with WinDom functions

AES	Windom	Comments
appl_init()	ApplInit()	incompatible
appl_exit()	ApplExit()	incompatible
appl_write()	ApplWrite()	compatible
(*)appl_control()	ApplControl()	compatible
others ...	no equivalent	
evnt_multi()	EvntWindom()	possible but without automatic management of AES events.
others ...	no equivalent	same remark
menu_bar()	MenuBar()	incompatible
menu_tnormal()	MenuTnormal()	incompatible
menu_icheck()	MenuIcheck()	incompatible
menu_text()	MenuText()	incompatible
others ...	no equivalent	
objc_change()	ObjcChange()	compatible
objc_draw()	ObjcDraw()	compatible
objc_edit()	ObjcEdit()	under construction
others ...	no equivalent	
form_alert()	no equivalent	
form_error()	no equivalent	
form_...	Form...	obsolets (see Form library)
wind_calc()	WindCalc()	compatible
wind_close()	WindClose()	incompatible
wind_create()	WindCreate()	incompatible
wind_delete()	WindDelete()	incompatible
wind_find()	WindHandle()	compatible
wind_get()	WindGet()	incompatible
wind_new()	WindNew()	under construction
wind_open()	WindOpen()	incompatible
wind_set()	WindSet()	incompatible
wind_update()	no equivalent	
rsrc_load()	RsrcLoad()	incompatible
rsrc_free()	RsrcFree()	incompatible
others...()	no equivalent	

(*) function only available with Naes

Programming guideline of WinDom

More about GEM ...

GEM is a GUI ie a Graphical User Interface. it was developped by Digital Research during the eighteen. It was probably the first multitasking GUI with X-window. GEM was originaly developped on C/PM (an Digital Research operating system) on PC compatible computers. Then GEM was adapted to MS-DOS and DR-DOS, Atari-ST with GEMDOS and even MacIntosh Lisa ! Although PC-GEM and ST-GEM are compatible, there are some differences and the evolution of ST-GEM due to Atari Corp is different to PC-GEM. The last version of PC-GEM is GEM/3 (in 1989). The last version of ST-GEM is MultiTOS on Falcon computer (in 1993) and they are different. Now, the PC-GEM (bought by Caldera) is a free software.

GEM is divided in two parts :

- VDI (Virtual Device Interface),
- AES (Application Environment Service).

VDI is devoted to handle all graphical peripherics (screen, printer, graphic palette) and more. It uses drivers and offers many function to drawn graphics primitiv, display texte with font. AES is devoted to handle the user interface. It offers windows, formulars, menu, desktop and an event manager.

The next table list the different version of TOS, AES and GEMDOS. The convention used fir number version is: a x.0y version denotes a x.y version. For example, 1.06 is 1.6 and 1.62 is 1.62 !

Differents version of TOS, GEMDOS and AES

TOS	Date	GemDos	AES	Computer
1.00	11-20-1985	0.13	1.20	ST, STM and STF
	06-02-1986	-	-	(There are two versions)
1.02	04-22-1987	0.13	1.20	STF and Mega-ST
1.04	04-06-1989	0.15	1.30	STF, Stacy (Rainbow TOS)
1.06	?	?	?	STE (preversion)
1.62	01-01-1990	0.17	1.40	STE
2.02	?	?	?	STE
2.05	?	?	3.10	Mega-STE, Stylus (a)
2.06	11-14-1991	0.20	3.20	Mega-STE (floppy 1.44M), ST-Book
3.01	?	?	3.00	TT030, (floppy 720k)
3.05	12-05-1990	0.19	3.10	TT030, (floppy 1.44M)
3.06	09-24-1991	0.20	3.20	TT030, (final version)
4.01	?	?	3.31	Falcon030 (prototype without DSP)
4.02	?	?	3.40	Falcon030 (prototype)
4.04	03-08-1993	0.30	3.40	Falcon030
4.92	06-22-1993	0.30	4.10	Beta version of TOS 5.00
				(Falcon040)
4.97	?	?	?	??
MiNT (b)				
MultiTOS	-	-	4.00	All computers
MultiTOS	-	-	4.10	"
Geneva	-	-	4.10	"
Naes	-	-	4.10	"
Magic (c)				
same TOS version	11.02.97	3.19	3.99	version 5.11

(a) The Stylus TOS is a special version of TOS 1.04 including an extension to handle the pen : PenOS.

(b) MultiTOS, Naes and Geneva replace the AES part of your OS. Nvdi replace the VDI part of your OS. MiNT remplace the GEMDOS of your OS (For GEM, GEMDOS is the operating system as MS-DOS DR-DOS or C/PM). Notice that MultiTOS of Naes need MiNT to run. Geneva can run with or without MiNT.

(c) MagiC is a complet TOS including its own AES and GEMDOS part. Actually, it is not a TOS but a TOS compatible system.

Emulators and computers TOS compatible

Computer/Emulator	TOS version
Falcon Mark I, II and III	4.04
Hades 040	3.06
Hades 060	3.06
Milan 040	5.0x
Milan 060	?
Gemulator	?
TOS2WIN	?
STonX	see the TOS image used
MagicMac	see MagiC
MagicPC	see Magic
Falcon Centurbo II	7.00 (*)

(*) Centek uses the 7.00 TOS number version to designe their own extension of the 4.04 falcon system, but this choice is very strange and the TOS version number should not be used to test the Centurbo II presence (prefer the cookiejar).

For more information consult the internet site : http://ic.net/~tjh/computers/atari/

Programming guideline of WinDom

AES rectangle list

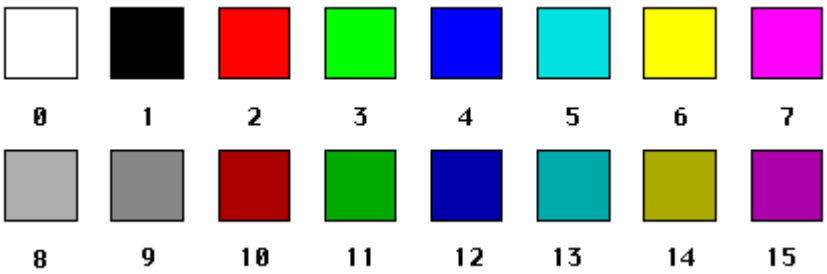
To redraw a window, AES use an algorithm based on a rectange list : each window area are a set of rectangle (in case of windows intersecting). This list is handled by AES and given by wind_set() with the mode WM_FIRSTXYWH and WM_NEXTXYWH.

Programming guideline of WinDom

Diverses

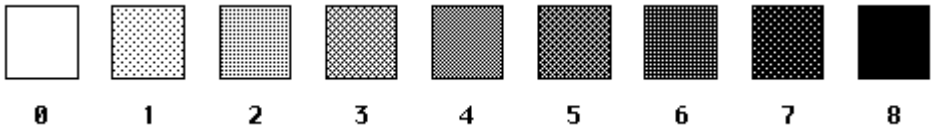
Programming guideline of WinDom

AES colors



Programming guideline of WinDom

AES style



Programming guideline of WinDom

VDI style pattern

Programming guideline of WinDom

VDI style hatched

Programming guideline of WinDom

BiG

BiG is Gem A GEM library written by Claude Attard

Programming guideline of WinDom

EgLib

EgLib is a GEM library written by Christophe Boyanique

Programming guideline of WinDom

Interface

Interface is a resource editor written by Olaf Meisiek.

Programming guideline of WinDom

MyDial

MyDial is a GEM library providing a collection of new object in formulars. written by ...

Programming guideline of WinDom

Let's them fly

Let's Them Fly, written by Oliver Scheel abd Darryl Pipper. e-mail: drpiper@cix.compulink.co.uk

Programming guideline of WinDom

ICFS

IConiFy Server is written by Dirk Haun.

Programming guideline of WinDom

Selectric

Selectric, a file selector written by Oliver Scheel.

Programming guideline of WinDom

Bubble GEM

Bubble GEM is written by Thomas Much. EMail: Thomas.Much@stud.uni-karlsruhe.de

Programming guideline of WinDom

untoppable

An untoppable window does not receive a **WM_TOPPED** message when a mouse button event occurs over its work area. A **MU_BUTTON** is sent. This feature allows bottomed windows to be used like topped windows.

Programming guideline of WinDom

WDK

Acronym for WinDom Developer Kit