rxwizenglish

COLLABORATORS

	TITLE : rxwizenglish		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 24, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rxwizenglish

1.1 Index

rxwiz.library 7.1

1.2 warning

```
THIS SOFTWARE AND INFORMATION ARE PROVIDED "AS IS".
ALL USE IS AT YOUR OWN RISK, AND NO LIABILITY OR
RESPONSIBILITY IS ASSUMED. NO WARRANTY IS MADE.
```

1.3 guide

```
    This guide contains links with inline ARexx macros wich
    use rmh.library/OpenURL. They will work if and only if
    you have both openurl.library and rmh.ibrary installed.

    rmh.library is by me and can be found on aminet at rexx/utils/rmh.lha
```

openURL.library is by Troels Walsted Hansen

"This library was created to make it easier for application programmers to include clickable URLs in their applications, about windows, etc. ..."

You can find it on aminet at comm/www/OpenURL20.lha

1.4 distribution

rxwiz.library is FreeWare.

You are free to detribute it as long as the original archive is kept intact.
Commercial use or its inclusion in other software package is prohibited ↩
without
prior consens from the Author.

1.5 author

I am Alfonso Ranieri .

My e-mail address is alfier@iol.it .

My home page is at http://users.iol.it/alfier/ .

You can find me on:
- #amigaita ircnet ;
- #amyita   ircnet .

1.6 introduction

It is a bridge between wizard.library and ARexx.

The gui is described out of the macro. In the macro you just
open the surfaces, the windows and handle them in an Amiga-Exec
way.

You must name any window and object you want to receive messages
from or you want to set/get attributes and you will be notified in
the macro with symbolic name.

StormWizard and wizard.library are H&P copyright.

StormWizard is the GUI maker for wizard.library and is a commercial
product. A demo of StormWizard is avaible free.

wizard.library is the AmigaDOS shared library that runs GUI created
by StormWizard; it is free distribuible.

This is a new version, compatible with old rxwiz.library, but
for the last time.

```
Function marked as "OBSOLETE" are supported in this version of
the library, but will be nomore in next version.

Now the library handles full GUI with appicon and commodity.

The main terms are:

event - an event that can occur on a gui e.g. an IDCMP
intuition message , an appicon message or a cx message.
rxwiz.library is AmigaOS exec signals event driven
so any event can be filtered via a signal.
Usually rxwiz.library handles particular events by itself,
but can be forced to not do that.

surface - it is the gui description, made of a wizard file
created by StormWizard, and the GUI description passed to
the library, in a macro, via "fields" of a "stem" .
In the GUI description you define if the GUI has an appicon
and a cx , if the GUI auto handles iconify - deiconify - close
events.

appicon - an AmigaOS appicon that is auto created if the GUI
has one, when the GUI is iconified

cx - a basic AmigaOS commodity with no hotkey (in this version)

windows - the windows defined in the wizard and opened in the
GUI

objects - the BOOPSI gadget defined in the wizard and opened
in a window
```

1.7 requirements

```
This library needs:
- AmigaOS, version >=3
- wizard.library, version >=38

If you want to create or to modify GUIs you need StormWizard not demo.

The examples require:
- rmh.library
- rxsocket.library
- rmh.library
- rxasl.library
```

1.8 installation

```
Run the installation script.
NOTA BENE: Catalogs are not installed by the installation script.
```

1.9 terms

```
- stem or stemName: a valid ARexx variable name e.g. var, var.0, var.name, var

- types of arguments: the types used are:
  D   any data            --
  N   int                 /N
  S   symbol              /S ARexx valid symbol
  V   stemName            /V ARexx valid symbol as S but with length<20
```

1.10 bugs

```
- no way for the library to know when a critical stack situition
  may appear and so self-arrange the stack, anyway critical stack
  situations appear only with very very big GUI (e.g. try to open
  AmigaWriter.wizard GUI window 4).
- hummmm maybe many others, let me know.
```

1.11 gui

```
To create a GUI you must use StormWizard, but not the demo version,
because it doesn’t let you save the GUI.

Name all the objects (menu items, gadgets, windows, ...) you want to
receive messages from or you want to get/set attributes.

Before saving the GUI, do a "Renumber all" for all the windows and
write down the last gadget ID.

Also write down the windows id.

You need these numbers to open your GUI.
```

1.12 functions

```
ActivateGadget
ActivateWindow
CheckMenu
CloseSurface
CloseWindow
GadgetConfig
GadgetKey
GetRxWizString
GetWindowAttr
GetWizAttr
HandleWindow
help
HelpControl
LockWindow
```

```
LockWindows
MenuStatus
OnMenu
OpenSurface
OpenWindow
SetStatcks
SetWindowattrs
SetWizattrs
snapshot
UnlockWindow
UnlockWindows
WindowSignal
WizEasyRequest
```

1.13 activategadget

```
ActivateGadget

Usage: res = ActivateGadget(surface,window,gadget)
<surface/S>,<window/S>,<gadget/S>

Activate a gadget.
Returns 0 for succes or a positive error code.
```

1.14 activatewindow

```
ActivateWindow

Usage: res = ActivateWindow(surface,window)
<surface/S>,<window/S>

Activate a window.
Returns 0 for succes or a positive error code.
```

1.15 checkmenu

```
CheckMenu

Usage: res = CheckMenu(surface,window,menu,status)
<surface/S>,<window/S>,<menu/S>,[status/N]

Check a menù item (if checkable).
menu is the name of the menu (you have to name it).
status is 0 or 1 (1 is the default)

If the menu can’t be found it returns 58  "object not found"
If the menu is not checkable it returns 61
"incomplente attribute reading or setting" .
```

1.16 closesurface

```
CloseSurface

Usage: res = CloseSurface(surface)
<surface/S>

Close a surface and all the windows that belong to it.

Returns 0 for succes or a positive error code.
```

1.17 closewindow

```
CloseWindow

Usage: res = CloseWindow(surface,win,ONLYWIN)
<surface/S>,<win/S>,[ONLYWIN]

Close a window opened with OpenWindow().
Surface is the name of the surface the window belongs to.
If ONLYWIN is given as third argument, only the window is
closed. It let you close the window, but still use the
gadgets in it, to set/get their attribute.
It also let you re-open the window very fast, without re-set
its gadgets.

Returns 0 for succes or a positive error code.
```

1.18 gadgetconfig

```
GadgetConfig

Usage: res = GadgetConfig(surface,win,obj,var)
<surface/S>,<win/S>,<obj/S>,<var/S>

Write in var the configuration string of a gadget in a window
of a surface as defined in the surface description in StormWizard.

Returns 0 for succes or a positive error code.
```

1.19 gadgetkey

```
GadgetKey

Usage: res=GadgetKey(surface,window,code,qualifier)
<surface/S>,<window/S>,<code/N>,<qualifier/N>

Notifies gadgets from a window of a surface of a key pressed
of code and qualifier specified.
```

```
If some gadget reserved that qualifier/code key combination,
it will notify its attributes via ICMPUPDATE.

Returns an ARexx boolean.
```

1.20 getrxwizstring

```
GetRxWizString

Usage: string=GetRXWizString(code)
<code/N>

Return a localized error string from an error code.
All the functions returns 0 for succes or a positive
integer for failure.
The string is localized; if the function is able to
open the catalog rxwiz.catalog" the string is read
from there. Internal library language is english.

Codes and strings actually defined are:
- 51   "too few memory"
- 52   "required argument missing"
- 53   "bad number"
- 54   "bad value"
- 55   "screen not found"
- 56   "surface not found"
- 57   "can't open window"
- 58   "object not found"
- 59   "object name already in use"
- 60   "object not created"
- 61   "incomplente attribute reading or setting"
- 62   "invalid hotkey description"

- 1000 "Can't find"
- 1001 "field"
- 1002 "line"
- 1003 "unknown error %ld" - (internal, DO NOT USE !!!)
```

1.21 getsurfaceattr

```
GetSurfaceAttr

Usage: res = GetSurfaceAttr(surface,attr,var)
<surface/S>,<attr/S>,<var/V>

Reads ONE attribute of a surface.
The value of the attribute is stored in var, in attribute specific way.
The attributes that can be read are:
Snaphost

Returns 0 for succes or a positive error code.
```

1.22 getwindowattr

```
GetWindowAttr

Usage: res = GetWindowAttr(surface,win,attr,var)
<surface/S>,<win/S>,<attr/S>,<var/V>

Reads ONE attribute of a window of a surface.
The value of the attribute is stored in var, in attribute specific way.
See attributes for a list of the attributes.

Returns 0 for succes or a positive error code.
```

1.23 getwizattr

```
GetWizAttr

Usage: res = GetWizAttr(surface,win,obj,attr,var)
<surface/S>,<win/S>,<obj/V>,<attr/S>,<var/V>

Reads ONE attribute from an obj in a window of a surface.
The value of the attribute is stored in var, in attribute specific way.
See Object Attributes for a list of the attributes.

Returns 0 for succes or a positive error code.
```

1.24 handlewindow

```
HandleWindow OBSOLETE

Usage: res = HandleWindow(surface,win,handle)
<surface/S>,<win/S>,[handleStem/V]

Handle windows of a surface.

- surface is the name of a surface
- win see below
- handle is a stem to pass arguments and where functions return results;
  the fields that may be passed are:
  WAIT      doesn’t return untill a signal or an intuition message cames
            (default 1)
  SIGNALS   signals to wait for
  CTRLC     wait for ctrl-c (2**12)
  MULTI     win is a stem not a window name

If the field MULTI of the stem handle is set, win is a list
of n windows names, win.0, win.1,... till win.n exists; all those
windows are handled. The result fields are store in handle.winName.i,
where winName is the name of the window and i is the index for the
i-th intuition message.

If the field MULTI of the stem win is not set, win is a window name,
and just that window is handled. The result field are store in
handle.i where i is the index for the i-th intuition message.

All the intuition messages pending on the window(s) are received.

The fields set by the function in the stem handle are:

- SIGNALS     the signal received (including window(s) signal(s))
- IMSG        the numebr of intuition messages received
- m.CLASS     the class of the m-th intuition message (in human form, or " ↩
   UNKNOWN")
- m.QUALIFIER the qualifier of the m-th intuition message
- m.CODE      the code of the m-th intuition message
- m.IADDR     the IAddr of the m-th intuition message (DON'T PLAY WITH IT!)
- m.WINDOW    the window name of the window receiving the message as defined
              in OpenWindow()
- m.OBJECTID  the object name as defined in the surface from wich the
              message comes
- m.COORD     only for HIERARCHY objects: the x.y.x.w... coord or the item
- m.CONFIG    the configuration string as defined in the surface
- m.HELP      the help string as defined in the surface if the class is  ↩
   GADGETHELP
- m.NUMTAGS   most of the objects notify the window with a IDCMPUPDATE  ↩
   intuition
              message and a taglist with attributes
              if m.CLASS is "IDCMPUPDATE" in this field there is the total  ↩
                 number
              of the tags. Th name of the tags are in m.tag.j,...,m.tag.x
              with x=m.NumTags-1. Any tags value is in m.TAgName.
              Seems so complicated, but it is very easy.

Returns 0 for succes or a positive error code.
```

1.25 handlesurface

```
HandleSurface

Usage: res = HandleSurface(surface,handle,win)
<surface/S>,<handleStem/V>,[win/S]

Handle a surface.

surface - the name of a surface
handle - a stem to pass arguments and where functions return results;
         the fields that may be passed are:
         wait - doesn't return untill a signal or an intuition message cames
                (default 1)
         signals -  signals to wait for
         ctrlc - wait for ctrl-c (2**12)
         multi - win is a stem not a window name
win - if present only this win is handled

The fields set by the function in the stem handle are:

- signals     the signal received (including window(s) signal(s))
- imsg        the number of messages received
- m.class     the class of the message (in human form, or "UNKNOWN")
              the class is an IDCMP name or:
              APPWINDOW - an appwindow message
              APPICON - an appicon message
              CXHOTKEY - hotkey was used (if hostkey defined)
              CXDISABLE - cx was disabled in Exchange
              CXUNABLE - cx was unabled in Exchange
              CXKILL - cx was removed in Exchange
              CXUNIQUE - another cx with the same name was opened (if defined ↩
                 and required)
              CXAPPEAR - "Show interface" was pressed in Exchange
              CXDISAPPEAR - "Hide interface" was pressed in Exchange
- m.qualifier the qualifier of the m-th intuition message
- m.code      the code of the m-th intuition message
- m.iaddr     the IAddr of the m-th intuition message (DON'T PLAY WITH IT!)
- m.window    the window name of the window receiving the message as defined
              in OpenWindow()
- m.objectid  the object name as defined in the surface from wich the
              message comes
- m.coord     only for HIERARCHY objects: the x.y.x.w... coord or the item
- m.config    the configuration string as defined in the surface
- m.help      the help string as defined in the surface if the class is  ↩
   GADGETHELP
- m.numtags   most of the objects notify the window with a IDCMPUPDATE  ↩
   intuition
              message and a taglist with attributes
              if m.CLASS is "IDCMPUPDATE" in this field there is the total  ↩
                 number
              of the tags. Th name of the tags are in m.tag.j,...,m.tag.x
              with x=m.NumTags-1. Any tags value is in m.TAgName.
              Seems so complicated, but it is very easy.
- m.AILock    a boolean only valid if class is APPICON or APPWINDOW and the  ↩
   objectid
              is not empty : the object id as a lock

Returns 0 for succes or a positive error code.
```

1.26 help

```
help

Usage: helpString=help(funName)
<funName>

Returns the arguments mask string of the rxwiz.library function "funName".
```

1.27 helpcontrol

```
HelpControl

Usage: res = HelpControl(surface,win,0|1)
```

```
<surface/S>,<win/S>,<0|1>

Switch on/off the ability of a window from a surface to receive
GADGETHELP intuition messages.

Returns 0 for succes or a positive error code.
```

1.28 lockwindow

```
LockWindow

Usage: res=LockWindow(surface,win)
<surface/S>,<win/S>

Lock the window.

Returns 0 for succes or a positive error code.
```

1.29 lockwindows

```
LockWindows

Usage: res=LockWindows(surface)
<surface/S>

Lock all the windows of a surface.

Returns 0 for succes or a positive error code.
```

1.30 menustatus

```
MenuStatus

Usage: call MenuStatus(surface,win,status)
<surface/S>,<win/S>,<status/N>

Set or clear the menu strip of a window of a surface.

Returns 0 for succes or a positive error code.
```

1.31 onmenu

```
OnMenu

Usage: res = OnMenu(surface,window,menu,status)
<surface/S>,<window/S>,<menu/S>,[status/N]
```

```
Able/Disable a menù.
menu is the name of the menu (yes you have to name it).
status is 0 or 1 (1 is the default)

If the menu can’t be found it returns 58  "object not found"
```

1.32 opensurface

```
OpenSurface

Usage: res=OpenSurface(surfaceName,objName,stem)
<surfaceName>,<objName/V>,[stem/V]

Opens a surface and add it in the macro with objName.
The surface is a wizard file created by StormWizard.
objName is also a stem name where are definied the fields
for the surface to open; if stem is passed the stem are read
from there.
The fields are:
Catalog - the catalog to open (critical: use the cd file created
          by StormWizard and with catcomp make the catalog with
          NOOPTIM option)
PubScreenName - the default pubscreen for the windows of this surface
FallBack - fallback to default pubscreen if the above does not exist
           default 1
Snapshot - snaphost all the windows of the surface when it is closed
           default 0
AppName - the default name of the appicon (if any) and the cx (if any)
Iconified - the initial status of the GUI; all window not "NoAutoClose"
            are iconified and and an appicon is opened (iy any)
            default 0
MasterWin - the name of a window ; when that window is closed
            the macro is halted
AutoClose - when the cx (if any) receives a KILL (users press "Remove"
            in Exchange) the macro is halted
            default 0
AutoIconify - when the cx (if any) receives a APPEAR (users press "Show
              Interface" in Exchange) the GUI is deiconified
              default 1
AppIcon - then name of a info file (without ".info" to be used
          as appicon; if it can be found or is "" the system tools
          default icon is used.
          this stem also add an appicon to the GUI
          default no appicon
AppIconName - overwrite AppName
CxTitle - the title of the cx, if present CxDescr must be present
CxDescr - the description of the cx, if present CxTitle must be present
CxHotKey - a valid HotKey description
CxFlags - on or more of "UNIQUE" and "NOTIFY"

Returns 0 for succes or a positive error code.

If you try to create a cx that was opened with the UNIQUE flags, an error
59  ("object name already in use") is returned.
```

```
If you try to create a cx with an invalid CxHotKey, an error
62 ("invalid hotkey description") is returned.

BUG:
The functions does not return error 62 if an invalid CxHotKey is
specified (Humm really don’t know why, will be maybe fixed in
next version).
```

1.33 openwindow

```
OpenWindow

Usage: res=OpenWindow(surface,objName,stem)
<surface/S>,<objName/V>,[stem/V]

Opens a specific window defined in the surface and add it in
the macro environment with objName.
objName is also a stem name where are definied the fields
for the window to open; if stem is passed the stem are read from there.

You can specify which window is to open in 3 ways:
- specifyng the id of the window as defined in StormWizard in the
  field "ID"
- specifyng the name of the window as defined in StormWizard in the
  field "WindowName"
- opening the window with objName as the name of the window as defing
  in StormWizard.

The only needed field to set in stem is:
Gads - the num of the gadgets of the window: must be the highest
       gadget id of the window gadgets  from StormWizard plus 1.

Other important fields are:
AppWindow - icon can be dropped in the window, that will receive
            a handle message of class APPWINDOW
NoWindow - the window is not open but only initialized
           any object in the window can be used
NoAutoOpen - this window is not to be opened when the gui is
             deiconified
NoAutoClose - this window is not to be iconified when the gui is

All the other window attributes are optional.

Returns 0 for succes or a positive error code.
```

1.34 setstacks

```
SetStacks

Usage: res=SetStacks(surface,dStack,sStack)
<surface/S>,[dStack/N],[sStack/N]
```

```
Due the way windows are drawn and attributes of objects of windows
are set, there may happen that crtitical stacks sitution occurs.
This functions set the dStack (the stack used for the drawing process
of wizard.library) and sStack (the stack used by rxwiz in stack swap).

Returns 0 for succes or a positive error code.
```

1.35 setsurfaceattrs

```
SetSurfaceAttrs{plain}

Usage: res = SetSurfaceAttrs(surface,stem)
<surface/S>,[stem/V]

Sets attributes" of a surface.
The attribute that can be set are:
Snapshot - snapshot the surface when it is closed
Iconified - iconfy the surface
AppIconName - new name of the app icon

Returns 0 for succes or a positive error code.
```

1.36 setwindowattrs

```
SetWindowAttrs

Usage: res = SetWindowAttrs(surface,win,stem)
<surface/S>,<win/V>,[stem/V]

Sets attributes of a window.
If given the attributes are read from stem, otherwise from win.

Returns 0 for succes or a positive error code.
```

1.37 setwizattrs

```
SetWizAttrs

Usage: res = SetWizAttrs(surface,win,obj,stem)
<surface/S>,<win/S>,<obj/S>,<stem/V>

Set attributes from an object in a window from a surface.
You must set stem with the fields of the attributes to set.
See Object Attributes for a list of the attribute.

Returns 0 for succes or a positive error code.
```

1.38 snapshot

```
snapshot

Usage: res=SnapShot(surface)
<surface/S>

Snapshot the surface.
Snapshot applies only on closed windows.

Returns 0 for succes or a positive error code.
```

1.39 unlockwindow

```
UnlockWindow

Usage: res=UnlockWindow(surface,win)
<surface/S>,<win/S>

Unlock a window locked by LockWindow().
If the window was not locked, just does nothing.

Returns 0 or a positive error code.
```

1.40 unlockwindows

```
UnlockWindows

Usage: res=UnlockWindows(surface)
<surface/S>

Unlock all the windows of a surface locked by LockWindows().

Returns 0 or a positive error code.
```

1.41 surfacesignal

```
SurfaceSignal

Usage: sig=SurfaceSignal(surface)
<surface/S>

Returns the signal of a surface.
The signal to Wait for is sig.

Any event on this surface can be trapped waiting for this
signal with rmh.library/Wait .
```

1.42 windowsignal

```
WindowSignal OBSOLETE

Usage: sig=WindowSignal(surface,win)
<surface/S>,<win/S>

Returns the window Intuition port signal.
The signal to Wait for is sig.

(WARNING: IN THIS VERSION OF THE LIBRARY ALL THE WINDOWS OF A SURFACE
HAS THE SAME SIGNAL)
```

1.43 wizeasyrequest

```
WizEasyRequest

Usage: res=WizEasyRequest(surface,win,reqID,var,text)
<surface/S>,<win/S>,<reqID/N>,[var/S],[text]

Open the wizard requester with id 'reqID'.
The requester is open in the same window screen.
The window is locked.
The gadget number pressed by the user is returned in 'var' is specified;
last right gadget has value 0.
Text is an optional text used iff you put a '%s' in the text of the
requester in StormWizard.
This function is critical: bad text set in StormWizard can cause a
lot of troubles.

DON'T USE MORE THAN ONE '%s' IN THE REQUESTER TEXT IN StormWizard.

Returns 0 for succes or a positive error code.
```

1.44 guidelines

```
I suggest to:
- to set NoAutoClose in a window if it is
  really required, e.g. the window is a popup window
  that appears for special events
- always leave the library to handle iconify deiconify
- always set AutoClose in the surface and trap the halt
  in the macro via a
  ...
  signal on halt
  ...
  halt:
  exit
  ...
- define a Master window and trap the halt as above

The suggested cycle for the gui is
```

```
you only have the gui to handle:
...
handle.wait = 1
res = HandleSurface(name,"HANDLE")
if res~=0 then call error(res)
do i=0 to handle.imsg-1
    select
        when handle.i.class== ...
        ...
    end
end
...

you have to handle other signals driven events
...
ss=SurfaceSignal(name)
recv=wait(or(other,ss))
if and(recv,ss)~=0 then call handle()
...

handle:
handle.wait = 0
res = HandleSurface(name,"HANDLE")
if res~=0 then call error(res)
do i=0 to handle.imsg-1
    select
        when handle.i.class== ...
        ...
    end
end
...
```

1.45 object attributes

```
Type is the type of the attribute:
- N numeric
- B boolean
- D any data
- V stem

Action tells in which contest the attribute can be use:
- N the attribute is notified by the object via IDCMPUPDATE
- S the attribute can be set via SetRxWizAttrs()
- G the attribute can be read via GetRxWisAttr()

    Name                      Type    Action
--------------------------------------------

    All gadgets
    - GAID                      N       N
    - CLASSID                   N       G
    - DISABLED                  B       SG
    - PRIORITY                  N       G
    - MINWIDTH                  N       G
    - MINHEIGHT                 N       G

    VGroup, HGroup
    - PAGE                      N       NSG

    Button
    - LABEL                     D       S

    String
    STRING                      D       NSG

    ChecBox
    - CHECKED                   B       NSG

    MX
    - ACTIVE                    N       NSG

    Label
    - LINES                     D       G

    Integer
    - INTEGER                   N       NSG

    Sroller
    - TOP                       N       NSG
    - VISIBLE                   N       SG
    - TOTAL                     N       SG

    Arrow
    - STEP                      N       S

    Listview, Multilistview
    - TOP                       N       SG
    - SELECTED                  N       NSG
    - DOUBLECLICK               B       N
    - LIST                      V       S
    - VISIBLE                   N       S

    Toggle
    - CHECKED                   N       NSG
    - LABEL                     D       SG

    Colorfield
    - PEN                       N       S

    Args
    - TEXT                      D       S
    - PEN                       N       S

    Gauge
    - TOTAL                     N       S
    - CURRENT                   N       S
    - FORMAT                    N       S

    Cycle
    - ACTIVE                    N       NSG
    - LABELS                    D       S
```

```
    Date
    - DAY                         N       SG
    - MONTH                       N       S
    - YEAR                        N       S

    Imagetoggle
    - CHECKED                     N       NSG

    ImagePopUp
    - SELECTED                    B       N
    - LABELS                      D       S

    TextPopup
    - SELECTED                    B       N
    - LABELS                      D       S

    VectorPopup
    - SELECTED                    B       N
    - LABELS                      D       S

    Palette
    - SELECTED                    N       NSG

    Hierarchy
    - IMAGEWIDTH                  N       S
    - TOP                         N       SG
    - LIST                        V       S
    - VISIBLE                     N       S
    - SELECTED                    N       NSG
    - DOUBLECLICK                 B       N

    Slider
    - LEVEL                       N       NSG
    - MIN                         N       SG
    - MAX                         N       SG

    StringField
    - TEXT                        D       NSG


Note:

- LABELS for Cycle, TextPopup, VectorPopup and ImagePopup
  is a string of labels separated by a newline ( d2c(10) "A"x )
  e.g. "One" || "A"x || "Two"

LIST
- LISTVIEWLIST
  it is a stem, from where are read all the fields in the form
  .0,...,.n  untill .n exists,
  .i.Sel              is a flag that tells the library if the node
                      is selected
  Let's suppose the name of this stem is "LIST";
  in LIST you can set the fields:
```

```
  - COLUMNS/N   the number of columns in the listview
  - DIVIDE/B    Must divide the field with a vertical line?
  - HEADER      the header that will be drawn at the beginning
                of the list in bold style

  Each single field can have set the fileds:
  - PEN
  - SPEN
  - STYLE           NORMAL UNDERLINED BOLD ITALIC
  - JUSTIFICATION   one of LEFT RIGHT CENTER

  The content of each node is a string where the special
  character | (ascii 124) is used to split the different
  columns.

  EXAMPLE
  list.Columns=2
  list.header = "This is|a listview"
  list.0="One|1";list.0.style="UNDERLINED BOLD"
  list.1="Two|2";list.1.pen=1
  set.LISTVIEWLIST="LIST"
  res=SetWizAttrs("SURFACE","WINDOW","LISTOBJECT","SET")

HIEARARCHY
- HIERARCHYLIST
  it is a stem, from where are read all the fields in the form
  .0,...n until .n exists
  .i.List (i=0,...,n) is a flag that tells the library if the node
                      is a sub tree; if so, it’s nodes are read from
                      the new stem .i
  .i.Sel              is a flag that tells the library if the node
                      is opened

  The same fields of LISTVIEWLIST are used.
```

1.46 Window Attributes

```
Attributes that can be passed to OpenWindow(), SetWindowAttrs() and ↩
   GetWindowAttr().

Type is the type of the attribute:
- N numeric
- B boolean
- D any data
- V stem

Action tells in which contest the attribute can be use:
- C the attribute can be used when the window is created
- G the attribute can be read via GetRxWindowAttr()
- S the attribute can be set via SetRxWindowAttrs()

    Name                      Type    Action
--------------------------------------------
    - ID                       N      C          see OpenWindow
    - WinName                  D      C                          ’’
```

- gads	N	C	''
- AppWindow	B	C	''
- NoAutOopen	B	CS	''
- NoAutoClose	B	CS	''
- NoWindow	B	C	''
- pubscreen	D	C	
- fallback	B	C	fallback to surface pubscreen
- title	D	CSG	
- idcmp	D	CSG	see IDCMP
- left	N	CSG	
- top	N	CSG	
- width	N	CSG	
- height	N	CSG	
- MinWidth	N	CSG	
- MinHeight	N	CSG	
- MaxWidth	N	CSG	
- MaxHeight	N	CSG	
- DetailPen	N	C	
- BlockPen	N	C	
- ScreenTitle	D	CSG	
- sizegadget	B	C	
- dragbar	B	C	
- depthgadget	B	C	
- closegadget	B	C	
- sizebright	B	C	
- sizebbottom	B	C	
- smartrefresh	B	C	
- simplerefresh	B	C	
- backdrop	B	C	
- reportmouse	B	C	
- gimmezerozero	B	C	
- borderless	B	C	
- activate	B	C	
- rmbtrap	B	CS	
- nocarerefresh	B	C	
- newlookmenus	B	C	
- menuhelp	B	C	
- pointerdelay	B	C	
- busypointer	B	C	
- autoadjust	B	C	
- innerwidth	N	C	
- innerheight	N	C	
- mousequeue	N	C	
- rptqueue	N	C	
- notifydepth	B	C	
- helpgroup	N	C	
- help	B	S	

1.47 idcmp

```
They are received as a message class in string form.

They are passed to a window in 2 forms:
- numeric   just the value as an integer
- human     a string made of one or mare of the following words
```

```
                separeted by space(s)

    - SIZEVERIFY
    - NEWSIZE
    - REFRESHWINDOW
    - MOUSEBUTTONS
    - MOUSEMOVE
    - GADGETDOWN
    - GADGETUP
    - REQSET
    - MENUPICK
    - CLOSEWINDOW
    - RAWKEY
    - REQVERIFY
    - REQCLEAR
    - MENUVERIFY
    - NEWPREFS
    - DISKINSERTED
    - DISKREMOVED
    - WBENCHMESSAGE
    - ACTIVEWINDOW
    - INACTIVEWINDOW"
    - DELTAMOVE
    - VANILLAKEY
    - INTUITICKS
    - IDCMPUPDATE
    - MENUHELP
    - CHANGEWINDOW
    - GADGETHELP
```

1.48 thanks

```
    Thanks to:
    - Andreas Kuerzinger for the German catalog
```