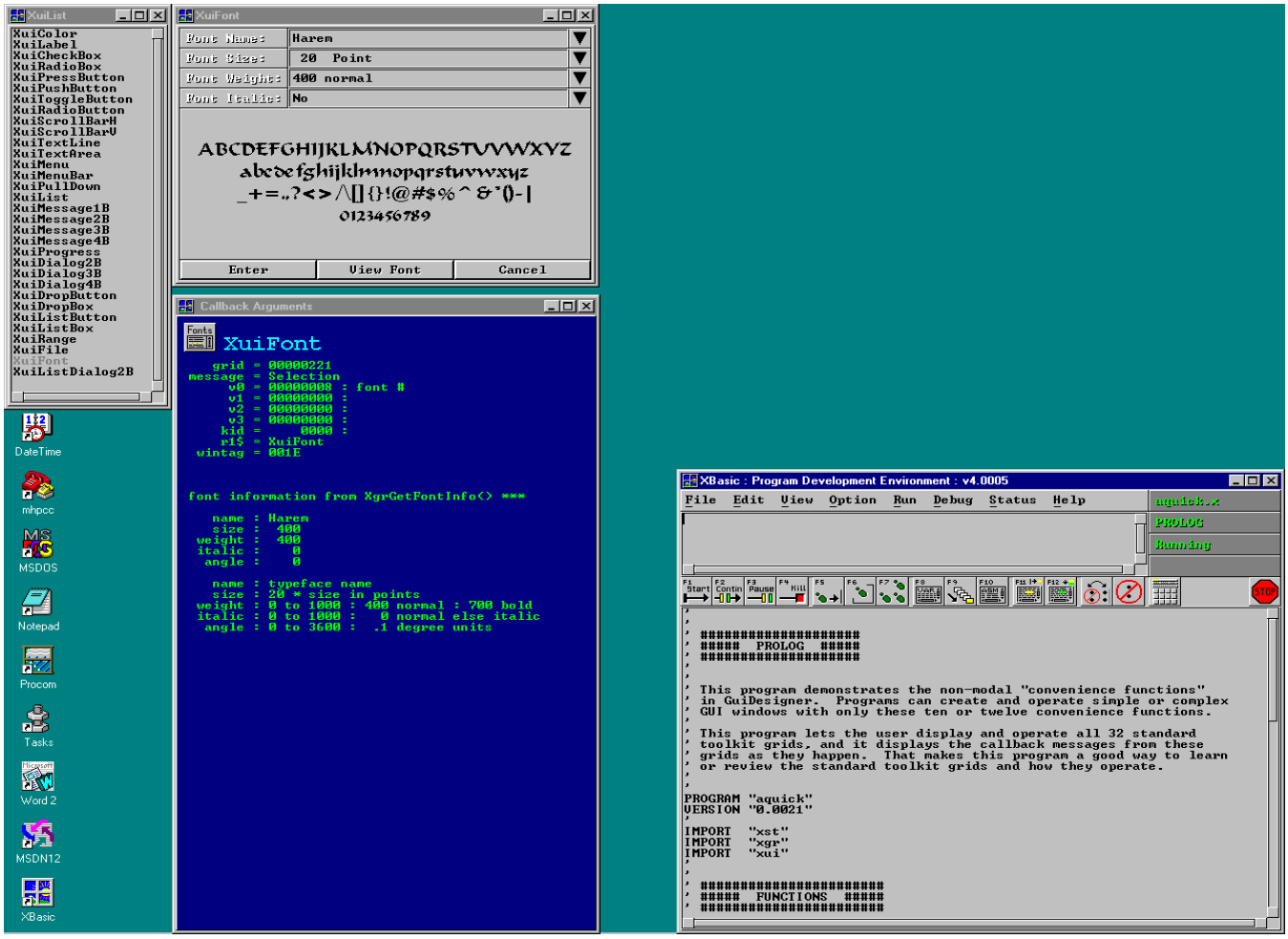
XBasic

Program Development Environment
(PDE)

GuiDesigner

Convenience Functions
Standard Toolkit Grids

Revision 0.0023
November 1, 1995



Sample program **`aquick.x`** *examines an* **`XuiFont`** *grid in action*

Terminology

If you have not read the *GraphicsDesigner* and *GuiDesigner* programmer guides, you may encounter unfamiliar terms in this manual. The following terms are important and often repeated in this document, and are therefore briefly described below:

grid short for graphics identity in *GraphicsDesigner* and *GuiDesigner*. To *GraphicsDesigner* a grid is a rectangular area in a window, and to *GuiDesigner* a grid is a label, button or any other GUI entity. The **`grid`** argument in *GraphicsDesigner* and *GuiDesigner* functions is the *grid number* that is assigned to each grid when it is created and uniquely identifies the grid. Almost every *GraphicsDesigner* and *GuiDesigner* function takes a grid argument to specify which grid the function refers to.

kid A kid, also known as a kid grid, is a grid within another grid, generally managed in some way by its *parent* grid. Each grid is identified by a *kid number* that is 0 for itself, followed by 1, 2, 3... for grids within itself, numbered from top to bottom, left to right. A grid can be specified by **`grid`**,**`kid`** in either of two ways:

`grid` = its grid # : **`kid`** = 0

`grid` = its parents grid # : **`kid`** = its kid # within its parent

GuiDesigner Convenience Functions

GuiDesigner Convenience Functions

In addition to a complete set of GUI development functions, *GuiDesigner* contains a small set of *convenience functions* that :

- create and operate modal windows with minimal code
- provide simplified GUI development methods for non-modal windows

The *GuiDesigner* convenience functions are :

```
XuiMessage ( message$ )
XuiDialog ( message$, default$, @kid, @reply$ )
XuiGetResponse ( gridType$, title$, message$, grid$, @v0, @v1, @kid, @reply$ )
XuiGetReply ( grid, title$, message$, grids$, @v0, @v1, @kid, @reply$ )

XuiCreateWindow ( @grid, gridType$, x, y, width, height, winType, display$ )
XuiSendStringMessage ( grid, message$, v0, v1, v2, v3, kid, r1 )
XuiQueueCallbacks ( grid, message, v0, v1, v2, v3, kid, r1 )
XuiGetNextCallback ( @grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$ )
```

Simple Input / Output

The two simplest modal convenience functions are essentially one-liner GUI equivalents of the **PRINT** statement and the **INLINE$()** intrinsic.

To display text:

```
PRINT "Hello Programmer"          ' console I/O
XuiMessage (@"Hello Programmer")  ' GUI equivalent
```

To display a prompt and input text:

```
reply$ = INLINE$ ("Input Your Name :  ")           ' console I/O
XuiDialog (@"Input Your Name", "", @kid, @reply$)  ' GUI equivalent
```

The final one-liner modal convenience function, **XuiGetResponse()**, will build and operate a modal window around any grid type.

Sophisticated GUI Programs

Add the other functions and you can create sophisticated GUI programs with any mix of modal and non-modal windows - with only 8 functions!

With the *GuiDesigner* convenience functions, you can develop programs with GUIs without designing your own windows. That's because the convenience functions will build a window around any grid type, whether from the standard toolkit, an accessory grid library, or any library your program **IMPORT**s.

The modal convenience functions will also build windows around grids defined by grid functions in your program. Those are windows you design interactively and have *GuiDesigner* convert to functions. Convenience function based GUI programs have few limitations, so you can develop capable GUI programs with nothing but convenience functions. The programs in the second part of this manual, plus sample programs **aquick.x** and **aeditors.x** are programs created with only convenience functions.

Non-Modal Windows

Most windows created by GUI programs are *non-modal windows*. Programs can create, configure, display, hide, and detect user actions in non-modal windows at any time, in any order. Whenever programs display more than one non-modal window at a time, the user is free to operate any of them. Programs can selectively display and hide non-modal windows to control the choices available to users.

Modal Windows

Modal windows appear, wait for a user response, then disappear. Modal windows "take over" an application until the user responds with a selection of some kind. Once displayed, modal windows intercept and discard all attempts to operate other windows created by the same program.

One-Liner Modal Windows

Three convenience functions create, configure, display, operate and destroy a modal window in one line:

```
XuiMessage ( message$ )
XuiDialog ( message$, default$, @kid, @reply$ )
XuiGetResponse ( gridType$, title$, message$, grid$, @v0, @v1, @kid, @reply$ )
```

Modal Windows with Convenience Functions

Three convenience functions let your programs create and configure any kind of window, then display, operate and hide it whenever they want :

```
XuiCreateWindow ( @grid, gridType$, x, y, width, height, winType, display$ )
XuiSendStringMessage ( grid, message$, v0, v1, v2, v3, kid, r1 )
XuiGetReply ( grid, title$, message$, grids$, @v0, @v1, @kid, @reply )
```

Non-Modal Windows with Convenience Functions

Two already mentioned plus two new convenience functions support simplified non-modal windows :

```
XuiCreateWindow ( @grid, gridType$, x, y, width, height, winType, display$ )
XuiSendStringMessage ( grid, message$, v0, v1, v2, v3, kid, r1 )
XuiQueueCallbacks ( grid, message, v0, v1, v2, v3, kid, r1 )
XuiGetNextCallback ( @grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$ )
```

Conventional Programs

The *GuiDesigner Programmer Guide* describes how to create *conventional GuiDesigner programs*, the framework for which is created by **FileNew GuiProgram** or by **.fn g** in the upper text area.

Conventional programs can call the modal convenience functions freely, but are not compatible with the non-modal convenience functions **XuiQueueCallbacks()** and **XuiGetNextCallback()**.

Convenience Programs

This manual shows how to write *GuiDesigner* convenience programs. The basic framework for these *convenience programs* is created by **FileNew Program** or by **.fn p** in the upper text area.

Convenience programs can call the modal and non-modal convenience functions freely, but non-modal convenience functions **XuiQueueCallbacks()** and **XuiGetNextCallback()** are not compatible with conventional *GuiDesigner* programs.

XuiMessage()

XuiMessage() is the simplest way to display a a modal window to display a message.

```
XuiMessage ( message$ )

  message$ - message to display on Label grid in XuiMessage1B
```

XuiMessage() displays a message in an **XuiMessage1B** grid and waits for the user to click the cancel button, press an Enter or Escape.

Newline characters ("\n") in the message string break the message into multiple lines. For example:

```
XuiMessage ( @"Hello Emperor\n\nDo You Realize\nYou Are Naked")
```

XuiMessage() *sample program*

```
'
' ####################
' #####  PROLOG  #####
' ####################
'
PROGRAM "naked"
VERSION "0.0000"
'
IMPORT  "xui"
'
DECLARE FUNCTION  Entry ()
'
'
' ######################
' #####  Entry ()  #####
' ######################
'
FUNCTION  Entry ()
'
  XuiMessage (@"Hello Emperor\n\nDo You Realize\nYou Are Naked")
'
END FUNCTION
END PROGRAM
```

XuiDialog()

XuiDialog() is the simplest way to display a modal window to display a message and input text.

```
XuiDialog ( message$, default$, @kid, @reply$ )

  message$ - display message on Label grid in XuiDialog2B
  default$ - display default input string in TextLine in XuiDialog2B
  kid      - kid # in XuiDialog2B that received the user response
  reply$   - contents of TextLine when XuiDialog2B got user response
```

XuiDialog() displays an **XuiDialog2B** grid with a message string on its **XuiLabel** grid and a default reply string in its **XuiTextLine** grid. Then it waits for the user to click the mouse on the "Enter" or "Cancel" button, or press an Enter or Escape key.

The kid # of the grid the user selected is returned in **kid**, along with the final value of the string in the **XuiTextLine** in **reply$**. If the user clicks the "Cancel" button or presses the Escape key, the reply string should be ignored (and may be returned empty).

XuiDialog() *sample program*

```
'
' ####################
' #####  PROLOG  #####
' ####################
'
PROGRAM "username"
VERSION "0.0000"
'
IMPORT  "xui"
'
DECLARE FUNCTION  Entry ()
'
'
' ######################
' #####  Entry ()  #####
' ######################
'
FUNCTION  Entry ()
'
  XuiDialog (@"Enter User Name", @"Bill Gates", @kid, @reply$)
'
  SELECT CASE kid
    CASE 1    : PRINT reply$; " : XuiLabel kid"
    CASE 2    : PRINT reply$; " : XuiTextLine kid"
    CASE 3    : PRINT reply$; " : XuiPushButton kid - \"Enter\""
    CASE 4    : PRINT reply$; " : XuiPushButton kid - \"Cancel\""
    CASE ELSE : PRINT reply$; " : Impossible kid #"; kid
  END SELECT
END FUNCTION
END PROGRAM
```

XuiGetResponse()

XuiGetResponse() is the simplest way to display a modal window that contains a grid of *arbitrary grid type*.

```
XuiGetResponse ( gridType$, title$, message$, grid$, @v0, @v1, @kid, @reply$ )

  gridType$ - grid type name (like "XuiFile", "XuiDialog3B", etc)
  title$    - display this string on the window title bar
  message$  - display this string on the 1st kid that's a Label grid
  grid$     - display these strings on kid grids following the Label grid
  v0,v1     - copies of v0,v1 arguments returned by the gridType$ grid
  kid       - kid # in gridType$ that received the user response
  reply$    - contents of textString or textArray as defined in gridType$
```

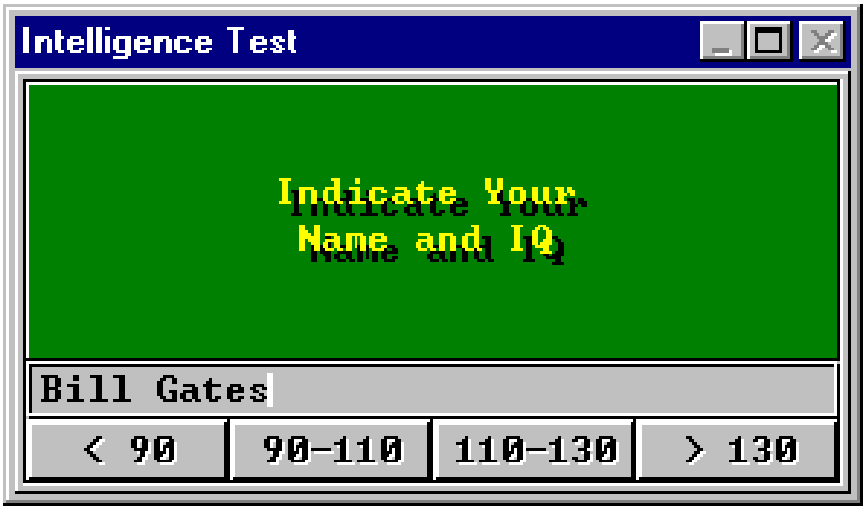
XuiGetResponse() displays a grid of the specified type in a modal window with a title string in the window title-bar and a message string in the first **XuiLabel** grid. It also changes the **TextString** property of any combination of kids following the first **XuiLabel** grid.

Newline characters in the message string break the text in the Label string into multiple lines. Newline characters in the grid string separate the text strings for each of the subsequent grids. Thus only the message label can be given a multiple line string.

The **kid** # of the grid the user selected is returned, along with the final value of the string from the kid given by the **inputTextString** or **inputTextArray** property of the grid type. If the grid type has both properties, the final string in the **inputTextString** kid is returned. Text arrays are returned as strings with "**\n**" between array elements.

If the user clicks the "Cancel" button or presses the Escape key, the reply string should be ignored (and may be returned empty).

XuiGetReponse() can create, display, and operate modal windows containing any grid type, including accessory grid types and grid types defined by grid functions in conventional *GuiDesigner* programs (convenience programs do not contain grid functions).



`XuiGetResponse()` *sample program*

```
'
' ####################
' #####  PROLOG  #####
' ####################
'
PROGRAM "smarts"
VERSION "0.0000"
'
IMPORT  "xui"
'
' silly program to demonstrate XuiGetResponse() function
'
DECLARE FUNCTION  Entry ()
'
'
' ######################
' #####  Entry ()  #####
' ######################
'
FUNCTION  Entry ()
'
  gridType$ = "XuiDialog4B"
  title$ = "Intelligence Test"
  message$ = "Indicate Your\nName and IQ"
  strings$ = "Bill Gates\n< 90\n90-110\n110-130\n> 130"
'
  XuiGetResponse (@gridType$, @title$, @message$, @strings$, @v0, @v1, @kid, @reply$)
'
  SELECT CASE kid
    CASE 2  : iq$ = "idiot"      ' entered name without selecting an IQ range - wise guy?
    CASE 3  : iq$ = "stupid"
    CASE 4  : iq$ = "normal"
    CASE 5  : iq$ = "smart"
    CASE 6  : iq$ = "liar"
  END SELECT
'
  XuiMessage ("*****   " + reply$ + "   *****\n\nIQ\nTest\nResults\n\n\"" + iq$ + "\"")
END FUNCTION
END PROGRAM
```

XuiGetReply()

XuiGetReply() is the simplest way to display a modal window that contains any grid type with *arbitrary properties*.

```
XuiGetReply ( grid, title$, message$, grid$, @v0, @v1, @kid, @reply$ )

  grid      - grid number of an already existing grid
  title$    - display this string on the window title bar
  message$  - display this string on the 1st kid that's a Label grid
  grid$     - display these strings on kid grids following the Label grid
  v0,v1     - copies of v0,v1 arguments returned by the gridType$ grid
  kid       - kid # in gridType$ that received the user response
  reply$    - contents of textString or textArray as defined in gridType$
```

Since the other modal convenience functions create and destroy each time they're called, it's impossible to specify properties beyond those defined by the function argument strings. There's no way to change the color of a kid grid, its font, or any other property. Perhaps most importantly, there's no way to assign grid names or help strings for the grid and its kids.

XuiGetReply() doesn't have these limitations because it does not create and destroy windows. Instead, your program creates windows in advance, before it calls **XuiGetReply()** - usually when they programs starts and are never destroyed. Your program can change grid properties in these windows whenever it wants.

The first argument to **XuiGetReply()** is the grid number returned by the function that created the window and grid, which is **XuiCreateWindow()** in convenience programs.

Except for the fact that you have to create and destroy the window/grid independently, **XuiGetReply()** operates just like **XuiMessage(), XuiDialog, XuiGetResponse()**.

`XuiGetReply()` *sample program*

```
'
' ####################
' #####  PROLOG  #####
' ####################
'
PROGRAM "leave"
VERSION "0.0000"
'
IMPORT  "xgr"
IMPORT  "xui"
'
DECLARE FUNCTION  Entry ()
'
'
' ######################
' #####  Entry ()  #####
' ######################
'
FUNCTION  Entry ()
'
  XuiCreateWindow      (@grid, @"XuiDialog4B", 200, 200, 128, 64, 0, "")
  XuiSendStringMessage ( grid, @"SetFont", 800, 600, 400, 0, 1, @"Serif")
  XuiSendStringMessage ( grid, @"SetTexture", $$TextureShadow, -1, -1, -1, 1, 0)
  XuiSendStringMessage ( grid, @"SetImage", 0, 0, 16, 16, 1, @"\\xb\\xxx\\xstop.bmp")
  XuiSendStringMessage ( grid, @"SetColorExtra", -1, -1, $$Black, $$Yellow, 1, 0)
  XuiSendStringMessage ( grid, @"SetColor", $$LightBlue, $$Yellow, -1, -1, 1, 0)
  XuiSendStringMessage ( grid, @"SetColor", $$Cyan, $$Yellow, -1, -1, 2, 0)
  XuiSendStringMessage ( grid, @"SetColor", $$BrightCyan, -1, -1, -1, 3, 0)
  XuiSendStringMessage ( grid, @"SetColor", $$BrightGreen, -1, -1, -1, 4, 0)
  XuiSendStringMessage ( grid, @"SetColor", $$Yellow, -1, -1, -1, 5, 0)
  XuiSendStringMessage ( grid, @"SetColor", $$BrightOrange, -1, -1, -1, 6, 0)
'
  title$ = "XuiGetReply ( )"
  message$ = "People Of Earth\n\nYou Must Leave"
  grids$ = "Immediately \nRight Now\nThis Minute\nToday\nManana"
'
  XuiGetReply (grid, @title$, @message$, @grids$, @v0, @v1, @kid, @reply$)
'
  SELECT CASE kid
    CASE 0    : PRINT reply$; " : XuiDialog4B grid"
    CASE 1    : PRINT reply$; " : XuiLabel kid"
    CASE 2    : PRINT reply$; " : XuiTextLine kid - \"Immediately\"
    CASE 3    : PRINT reply$; " : XuiPushButton kid - \"Right Now\""
    CASE 4    : PRINT reply$; " : XuiPushButton kid - \"This Minute\""
    CASE 5    : PRINT reply$; " : XuiPushButton kid - \"Today\""
    CASE 6    : PRINT reply$; " : XuiPushButton kid - \"Manana\""
    CASE ELSE : PRINT reply$; " : Impossible kid # ::: !!! Yikes !!!"
  END SELECT
END FUNCTION
END PROGRAM
```

Selection Modal Convenience Functions

All the modal convenience functions are appropriate for simple utilities, personal programs, and in-house applications. Only **`XuiGetReply()`** is supports the built-in *InstantHelp*, however, so **`XuiGetReply()`** is the only modal convenience function commercial applications call.

Custom Convenience Functions

You can easily make your own one-liner modal convenience functions. Just create a function with whatever arguments are necessary to configure your window. When your program calls it, it can :

create and configure a window based on the arguments
have **`XuiGetReply()`** display, operate, hide the modal window
extract information from the grid if necessary
setup return values or perform some action

The following example is a custom convenience function designed to display an **`XuiFile`** grid/window and input a filename from the user.

```
' ####################
' #####  PROLOG  #####  create and test sample modal convenience function GetFilename
' ####################
'
PROGRAM  "getfile"
VERSION  "0.0000"
'
IMPORT  "xst"
IMPORT  "xgr"
IMPORT  "xui"
'
DECLARE FUNCTION  Entry       ()
DECLARE FUNCTION  GetFilename (@filename$, @attributes)
'
' ######################
' #####  Entry ()  #####  test GetFilename() - a sample modal convenience function
' ######################
'
FUNCTION  Entry ()
'
  DO
    a$ = ""
    GetFilename (@file$, @attributes)
    IFZ file$ THEN PRINT "*****  Cancel  *****" : EXIT DO
'
    SELECT CASE ALL TRUE
      CASE (attributes AND $$FileReadOnly)    : a$ = a$ + "FileReadOnly "
      CASE (attributes AND $$FileHidden)      : a$ = a$ + "FileHidden "
      CASE (attributes AND $$FileSystem)      : a$ = a$ + "FileSystem "
      CASE (attributes AND $$FileDirectory)   : a$ = a$ + "FileDirectory "
      CASE (attributes AND $$FileArchive)     : a$ = a$ + "FileArchive "
      CASE (attributes AND $$FileNormal)      : a$ = a$ + "FileNormal "
      CASE (attributes AND $$FileTemporary)   : a$ = a$ + "FileTemporary "
      CASE (attributes AND $$FileAtomicWrite) : a$ = a$ + "FileAtomicWrite "
      CASE (attributes AND $$FileExecutable)  : a$ = a$ + "FileExecutable "
    END SELECT
'
    IFZ a$ THEN a$ = "FileNonexistent"
    PRINT "\""; file$; "\" : "; HEX$ (attributes,8); " = "; a$
  LOOP
END FUNCTION
'
' ############################
' #####  GetFilename ()  #####  sample convenience function to input full filename$
' ############################
'
FUNCTION  GetFilename (filename$, attributes)
  filename$ = ""
  attributes = $$FALSE
  XuiGetResponse (@"XuiFile", @"SelectFile", "", "", @v0, @v1, @kid, @filename$)
  IF (v0 = -1) THEN filename$ = ""
  IF filename$ THEN XstGetFileAttributes (@filename$, @attributes)
END FUNCTION
END PROGRAM
```

Non-Modal Convenience Functions

You can create complete sophisticated applications with any number of non-modal windows with convenience functions too. Create and configure windows in the same manner as you would prior to calling **`XuiGetReply()`**. But add a **`"SetCallback"`** line like the second line below :

```
  XuiCreateWindow      (@g, @"XuiColor", 100, 256, 0, 0, 0, "")
  XuiSendStringMessage ( g, @"SetCallback", g, &XuiQueueCallbacks(), -1, -1, $win, -1)
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 0, @"Color")
  XuiSendStringMessage ( g, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
  XuiSendStringMessage ( g, @"HideWindow", 0, 0, 0, 0, 0, 0)
```

`$win` is an integer you choose to distinguish this window from others. If your program has only one window, this argument can be **`-1`** or **`0`**.

Your programs have to explicitly display and hide non-modal windows. Your programs can send **`"DisplayWindow"`** and **`"HideWindow"`** messages at any time to display and hide windows.

Non-Modal Message Loop

Programs wait for callback messages from non-modal windows in a message loop like the one below. Most callback messages are initiated by user actions like mouse button clicks and Enter keystrokes.

```
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
SUB Callback
  win = kid >> 16      ' win = which window is this callback from
  kid = kid AND 0xFF   ' kid = which kid in the window is this callback from
  GOSUB @sub[win]      ' handler for this window. or @func[] or SELECT CASE win
END SUB
```

The message loop is actually two loops - the outer loop processes messages one by one, and the inner loop processes all callback messages received by **`XuiQueueCallbacks()`** from processing each one.

`XgrProcessMessages(1)` waits for a message to become available, processes it, then returns. Processing a message usually does not generate a callback message. When it does, it usually generates a single callback message, though it will occasionally generate two or three.

`XuiGetNextCallback()` checks to see if **`XuiQueueCallbacks()`** received any callback messages as a result of processing the message. If no callback message was received, **`XuiGetNextCallback()`** returns a zero which terminates the inner loop and drops back into the outer message processing loop. If **`XuiQueueCallbacks()`** has one or more callback messages waiting, **`XuiGetNextCallback()`** transfers the callback arguments to its own arguments and returns a non-zero value. This drops the code into **`GOSUB Callback`** which calls a subroutine that processes the callback message.

`XuiGetNextCallback()`

`XuiGetNextCallback()` returns the same callback arguments as conventional programs, except the upper 16-bits of **`kid`** contains the window identifier set by the **`"SetCallback"`** message, and **`r1$`** contains the **`GridName`** property of the grid that initiated the callback. Your program can respond to specific grids in a most convenient and readable manner if it sends **`"SetGridName"`** to every grid it creates, including kid grids. Most programs respond to callbacks from each window separately :

```
'
  XuiCreateWindow      (@g, @"XuiDialog2B", 20, 20, 256, 128, 0, "")
  XuiSendStringMessage ( g, @"SetCallback", g, &XuiQueueCallbacks(), -1, -1, $$WindowEmployee, -1)
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 0, @"Employee")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 1, @"EmployeeLabel")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 2, @"EmployeeName")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 3, @"EmployeeEnter")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 4, @"EmployeeCancel")
  XuiSendStringMessage ( g, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
  XuiCreateWindow      (@g, @"XuiDialog2B", 20, 172, 256, 128, 0, "")
  XuiSendStringMessage ( g, @"SetCallback", g, &XuiQueueCallbacks(), -1, -1, $$WindowCompany, -1)
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 0, @"Company")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 1, @"CompanyLabel")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 2, @"CompanyName")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 3, @"CompanyEnter")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 4, @"CompanyCancel")
  XuiSendStringMessage ( g, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
  XuiCreateWindow      (@g, @"XuiDialog2B", 20, 324, 256, 128, 0, "")
  XuiSendStringMessage ( g, @"SetCallback", g, &XuiQueueCallbacks(), -1, -1, $$WindowJob, -1)
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 0, @"Job")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 1, @"JobLabel")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 2, @"JobName")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 3, @"JobEnter")
  XuiSendStringMessage ( g, @"SetGridName", 0, 0, 0, 0, 4, @"JobCancel")
  XuiSendStringMessage ( g, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' *****  Callback  *****
'
SUB Callback
  win = kid >> 16          ' win = which window is this callback from
  kid = kid AND 0xFF       ' kid = which kid in the window is this callback from
  SELECT CASE win
    CASE $$WindowEmployee  : GOSUB WindowEmployee
    CASE $$WindowCompany   : GOSUB WindowCompany
    CASE $$WindowJob       : GOSUB WiddowJob
  END SELECT
END SUB
'
' *****  WindowEmployee  *****
'
SUB WindowEmployee
  SELECT CASE r1$
    CASE "EmployeeName"    : PRINT "XuiTextLine expecting employee name"
    CASE "EmployeeEnter"   : PRINT "XuiPushButton labeled Enter"
    CASE "EmployeeCancel"  : PRINT "XuiPushButton labeled Cancel"
  END SELECT
END SUB
'
' *****  WindowCompany  *****
'
SUB WindowCompany
  SELECT CASE r14
    CASE "CompanyName"     : PRINT "XuiTextLine expecting company name"
    CASE "CompanyEnter"    : PRINT "XuiPushButton labeled Enter"
    CASE "CompanyCancel"   : PRINT "XuiPushButton labeled Cancel"
  END SELECT
END SELECT
'
' *****  WindowJob  *****
'
SUB WindowJob
  SELECT CASE r14
    CASE "CompanyName"     : PRINT "XuiTextLine expecting company name"
    CASE "CompanyEnter"    : PRINT "XuiPushButton labeled Enter"
    CASE "CompanyCancel"   : PRINT "XuiPushButton labeled Cancel"
  END SELECT
END SUB
```

Are GUI Programs Really Different ???

Contrary to common "wisdom", GUI programs in general, and especially GUI programs constructed from convenience functions, are practically identical structurally to conventional non-GUI programs. Consider the following common code from a conventional program :

```
  DO
    GOSUB DisplayCommands                  ' prints a list of command strings
    a$ = INLINE$ ("enter command ===>> ")  ' wait for user input
    IF a$ THEN GOSUB Callback              ' if command received then process it
 LOOP
'
SUB Callback
  SELECT CASE a$
    CASE "Help"   : GOSUB Help             ' user typed "Help"
    CASE "Move"   : GOSUB Move             ' user typed "Move"
    CASE "Rotate" : GOSUB Rotate           ' user typed "Rotate"
    ....          : more commands          ' user typed commands
    CASE ELSE     : PRINT "Say What?"      ' user typed garbage
  END SELECT
END SUB
```

GUI programs do the same thing. They display a menu of commands, wait for a user reply, call a routine to process the command, then loop. The only difference is that the conventional program **PRINT**s the menu while the GUI program puts the commands on buttons or menu bars. And while the conventional program always inputs the command as a line of text, the GUI program can also respond to button clicks and menu selections. But in essence, both are structurally identical.

The equivalent program with convenience functions is something like :

```
  GOSUB DisplayCommands          ' displays a window with buttons or menubar or ???
  DO
    XgrProcessMessages (1)       ' wait for user input
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback             ' if command received, process it
    LOOP
  LOOP
  RETURN
'
SUB Callback
  SELECT CASE r1$
    CASE "Help"   : GOSUB Help                  ' grid name = "Help"
    CASE "Move"   : GOSUB Move                  ' grid name = "Move"
    CASE "Rotate" : GOSUB Rotate                ' grid name = "Rotate"
    ....          : more commands               ' grid name = ...
    CASE ELSE     : XuiMessage (@"Say What?")   ' not known
  END SELECT
END SUB
```

GOSUB DisplayCommands does the same thing different ways.
a$ = INLINE$() becomes **XgrProcessMessages (1)**.
The subroutine is the same.

And you thought GUI programs were radically "different".

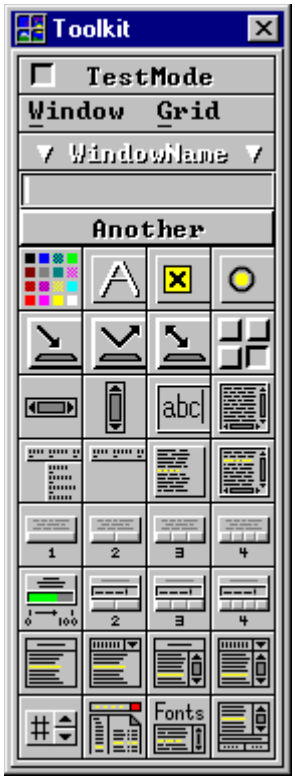
Think again!

GuiDesigner Standard Toolkit Grids

GuiDesigner Standard Toolkit

Click the toolkit hot button in the main window to display the toolkit in the lower left side of the display.

toolkit hot button



Standard Toolkit

standard grid types

XuiColor : Color grid (select one of the 125 standard *GraphicsDesigner* colors).

XuiLabel : Label grid (display single or multi-line message text and/or image).

XuiCheckBox : CheckBox grid (toggle on/off).

XuiRadioBox : RadioBox grid (toggle on turns all other RadioBox grids in group off).

XuiPressButton : PressButton grid (selection on MouseDown).

XuiPushButton : PushButton grid (selection on MouseDown - MouseUp).

XuiToggleButton : ToggleButton grid (toggle on/off on MouseDown).

XuiScrollBarH : ScrollBarH grid (horizontal motion / position control).

XuiScrollBarV : ScrollBarV grid (vertical motion / position control).

XuiTextLine : TextLine grid (one line text input / output).

XuiTextArea : TextArea grid (multi-line text input / output).

XuiMenu : Menu grid (create / manage pulldown list for MenuBar entries).

XuiMenuBar : MenuBar grid (select one of several horizontal entries).

XuiPullDown : Pulldown grid (select one of several vertical entries).

XuiList : List grid (scrollable list - select one of several vertical entries).

XuiMessage1B : Message box with 1, buttons.

XuiMessage2B : Message box with 2 buttons.

XuiMessage3B : Message box with 3 buttons.

XuiMessage4B : Message box with 4 buttons.

XuiProgress : Progress box.

XuiDialog2B : Dialog box with 2 buttons.

XuiDialog3B : Dialog box with 3 buttons.

XuiDialog4B : Dialog box with 4 buttons.

XuiDropButton : Button activated PullDown List.

XuiDropBox : PullDownList box.

XuiListButton : Button activated scrollable List.

XuiListBox : Scrollable List box.

XuiRange : Set value within a Range.

XuiFile : Select File Dialog.

XuiFont : Select Font Dialog.

XuiListDialog2B : Select List Entry Dialog with 2 buttons.

Standard Toolkit Grids

The rest of this document describes grid types in the standard toolkit. The information on each grid type includes an image of the grid in a window, a list of kids the grid contains, the callback messages and arguments that can be produced by grids of the specified grid type, and a short function that presents a grid of the specified grid type.

A complete program for each grid based on these functions is located in the **`\xb\app`** directory. **`\xb\aquick.x`** & **`\xb\atools.x`** are sample programs that demonstrates all 32 grids.

The convenience functions and standard toolkit grids are both described in this document because all the grid demonstration programs are convenience function programs.

Standard Toolkit Grids - sample programs

The complete sample program for each grid consists of :

- PROLOG with import statements and function declarations
- grid demonstration functions - **`Color()`**, **`Label()`**, **`CheckBox()`** ...
- a function that creates a window to display callback arguments
- a function that displays callback arguments in a generic way

The demonstration functions are slightly different for each grid. The PROLOG and support functions are the same for all grids, however, and are thus shown once on the following page and not repeated for every grid.

Each of the demonstration functions generally does the following:

1. create a window around a grid of the specified grid type
2. set its callback function
3. sets its grid name
4. display the window
5. wait in convenience function message loop for callback message
6. call a routine that displays the callback arguments

Callback Messages

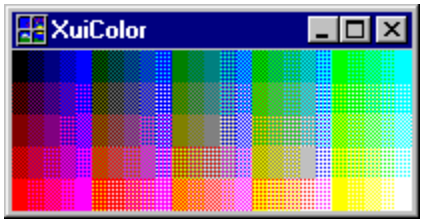
Most grids generate callback messages, but not all. For example, **`XuiLabel`** and **`XuiProgress`** grids are meant to display information but not respond to user actions like button clicks or keystrokes. Nonetheless, many grids pass on callbacks they receive but don't act upon, so even grids like **`XuiLabel`** and **`XuiProgress`** may generate callback messages like **`Help`** and **`CloseWindow`**.

Only callback messages initiated by each grid is noted in this document.

PROLOG and Support Functions

```
'
'
' ####################
' #####  PROLOG  #####
' ####################
'
PROGRAM "name"
VERSION "0.0000"
'
IMPORT  "xst"
IMPORT  "xgr"
IMPORT  "xui"
'
INTERNAL FUNCTION  GridName        ( )
INTERNAL FUNCTION  GetDisplayGrid  ( @label )
INTERNAL FUNCTION  DisplayCallback ( grid, message$, v0, v1, v2, v3, kid, r1$ )
'
'
' ##########################
' #####  GridName ()  #####  grid demonstration function goes here in complete program
' ##########################
'
FUNCTION  GridName ()
'
' function for each grid
'
END FUNCTION
'
'
' ###############################
' #####  GetDisplayGrid ()  #####
' ###############################
'
FUNCTION  GetDisplayGrid ( label )
  STATIC  grid
'
  IFZ grid THEN
    XuiCreateWindow      (@grid, @"XuiLabel", 100, 100, 512, 128, 0, "")
    XuiSendStringMessage ( grid, @"SetColor", $$BrightCyan, -1, -1, -1, 0, 0)
    XuiSendStringMessage ( grid, @"SetAlign", $$AlignUpperLeft, 0, 0, 0, 0, 0)
    XuiSendStringMessage ( grid, @"SetJustify", $$JustifyLeft, 0, 0, 0, 0, 0)
    XuiSendStringMessage ( grid, @"SetIndent", 6, 4, 0, 0, 0, 0)
    XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
  END IF
  label = grid
END FUNCTION
'
'
' ###############################
' #####  DisplayCallback ()  #####
' ###############################
'
FUNCTION  DisplayCallback ( grid, message$, v0, v1, v2, v3, kid, r1$ )
'
  XgrGetGridType (grid, @gridType)
  XgrGridTypeNumberToName (gridType, @gridType$)
  text$ = gridType$ + "\n"
  text$ = text$ + "    grid = " + HEX$ (grid,8) + "\n"
  text$ = text$ + " message = " + message$ + "\n"
  text$ = text$ + "      v0 = " + HEX$ (v0,8) + "\n"
  text$ = text$ + "      v1 = " + HEX$ (v1,8) + "\n"
  text$ = text$ + "      v2 = " + HEX$ (v2,8) + "\n"
  text$ = text$ + "      v3 = " + HEX$ (v3,8) + "\n"
  text$ = text$ + "     kid = " + HEX$ (kid,8) + "\n"
  text$ = text$ + "     r1$ = " + r1$
'
  GetDisplayGrid (@label)
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)
END FUNCTION
END PROGRAM
```

XuiColor




```
Kid 0 - XuiColor
  "Selection" callback on MouseDown
    v0 : color #           - 0 to 124
    v1 : red intensity     - 0x0000 to 0xFFFF
    v2 : green intensity   - 0x0000 to 0xFFFF
    v3 : blue intensity    - 0x0000 to 0xFFFF
```

XuiColor grids display all 125 *GraphicsDesigner* standard colors. The standard colors are numbered 0 to 124, with **$$Black = 0** in the upper left and **$$White = 124** in the lower right.

The *GraphicsDesigner* documentation tells how the colors are chosen and numbered. The color numbers in **XuiColor** grids increase from top to bottom : left to right, so the top row contains color #s **0** to **24**.

XuiColor grids contain no kids.

When the mouse is clicked on any color, an **XuiColor** grid generates a **Selection** callback message with arguments specifying the standard color number plus red, green, blue color intensities where each intensity can range from **0x0000** to **0xFFFF**.

XuiColor grids are currently not resizable.

```
'
'
' ###################
' #####  Color  #####
' ###################
'
FUNCTION  Color ()
  $XuiColor = 0
'
  GetDisplayGrid ( @label )
'
' create window with XuiColor grid
'
  XuiCreateWindow      (@grid, @"XuiColor", 100, 256, 0, 0, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiColor, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Color")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XgrColorNumberToName (v0, @color$)
  IF color$ THEN color$ = " : " + color$
  text$ = "XuiColor\n"
  text$ = text$ + "    grid : " + HEX$ (grid,8) + "\n"
  text$ = text$ + " message : " + message$ + "\n"
  text$ = text$ + "      v0 : " + HEX$ (v0,8) + " : standard color #" + color$ + "\n"
  text$ = text$ + "      v1 : " + HEX$ (v1,8) + " : red intensity\n"
  text$ = text$ + "      v2 : " + HEX$ (v2,8) + " : green intensity\n"
  text$ = text$ + "      v3 : " + HEX$ (v3,8) + " : blue intensity\n"
  text$ = text$ + "     kid : " + HEX$ (kid,8) + "\n"
  text$ = text$ + "     r1$ : " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
  XgrGetGridBox (label, @x1, @y1, @x2, @y2)
  XgrFillBox (label, v0, x2-174, y1+8, x2-8, y1+36)
END SUB
END FUNCTION
```

XuiLabel

Kid 0 - XuiLabel - does not initiate callback messages

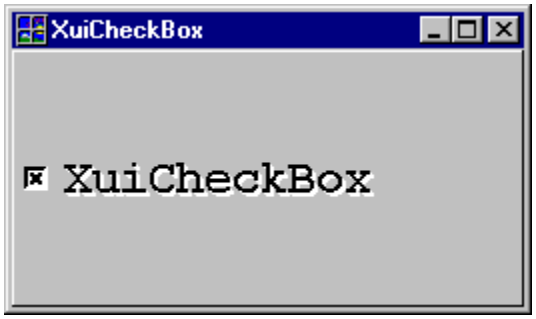
XuiLabel grids display information but do not initiate callback messages in response to user actions like keystrokes or mouse clicks.

The appearance of **XuiLabel** grids can be modified in many ways. The default appearance attributes of **XuiLabel** grids include:

- raised text
- empty text string
- slightly raised border

```
'
'
' ###################
' #####  Label  #####
' ###################
'
FUNCTION  Label ()
  STATIC  label
  $XuiLabel = 1
'
  GetDisplayGrid (@label)
'
' create window with XuiLabel grid
'
  XuiCreateWindow      (@grid, @"XuiLabel", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiLabel, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Label")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiLabel")
  XuiSendStringMessage ( grid, @"SetColor", $$BrightCyan, $$Yellow, -1, -1, 0, 0)
  XuiSendStringMessage ( grid, @"SetFont", 480, 700, 700, 0, 0, @"Roman")
  XuiSendStringMessage ( grid, @"SetTexture", $$TextureShadow, 0, 0, 0, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
END SUB
END FUNCTION
```

XuiCheckBox



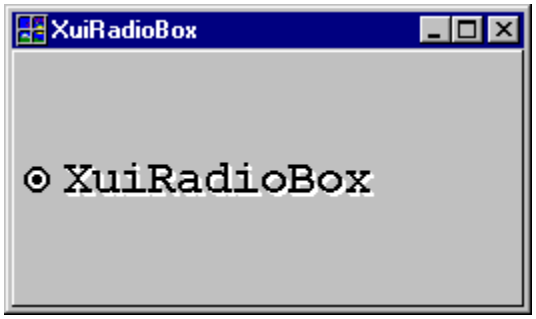

```
Kid 0 - XuiCheckBox
  "Selection" callback on Enter KeyDown
  "Selection" callback on MouseDown
     v0 : $$FALSE = not selected
          $$TRUE = selected
```

XuiCheckBox grids display a text string next to a small square that is either selected or not selected.

Selecting a **XuiCheckBox** grid toggles it to the opposite state.

```
'
'
' ######################
' #####  CheckBox  #####
' ######################
'
FUNCTION  CheckBox ()
  $XuiCheckBox = 2
'
  GetDisplayGrid ( @label )
'
' create window with XuiCheckBox grid
'
  XuiCreateWindow      (@grid, @"XuiCheckBox", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiCheckBox, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"CheckBox")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiCheckBox")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  IF v0 THEN selected$ = "selected" ELSE selected$ = "not selected"
  text$ = "XuiCheckBox\n\n "
  text$ = text$ + selected$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiRadioBox




```
Kid 0 - XuiRadioBox
  "Selection" callback on Enter KeyDown
  "Selection" callback on MouseDown
     v0 : $$FALSE = not selected
          $$TRUE = selected
```

XuiRadioBox grids display a text string next to a small circle that is either selected or not selected.

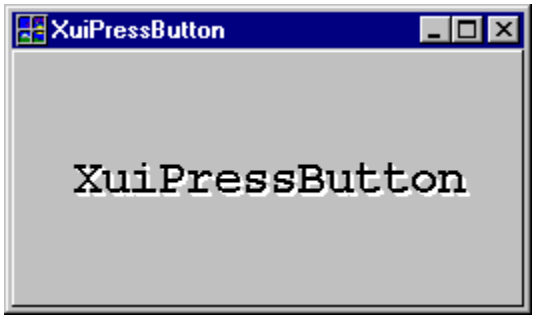
Selecting a **XuiRadioBox** grid first deselects all other **XuiRadioBox** grids in the same window and group, then selects itself. Therefore no more than one **XuiRadioBox** grid in a group is ever selected.

XuiRadioBox grids generate a **Selection** callback message with **v0=$$FALSE** when deselected and **v0=$$TRUE** when selected.

A window with a single **XuiRadioBox** is not particularly useful, since once selected, the only ay the user can deselect it is by selecting another **XuiRadioBox** grid in the window, none of which exist.

```
'
'
' ######################
' #####  RadioBox  #####
' ######################
'
FUNCTION  RadioBox ()
  $XuiRadioBox = 3
'
  GetDisplayGrid ( @label )
'
' create window with XuiRadioBox grid
'
  XuiCreateWindow      (@grid, @"XuiRadioBox", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiRadioBox, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"RadioBox")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiRadioBox")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  IF v0 THEN selected$ = "selected" ELSE selected$ = "not selected"
  text$ = "XuiRadioBox\n\n "
  text$ = text$ + selected$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiPressButton



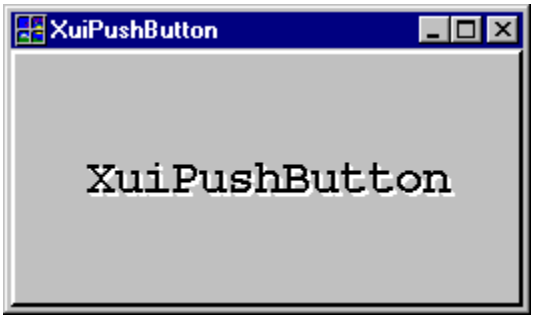

```
Kid 0 - XuiPressButton
  "Selection" callback on Enter KeyDown
  "Selection" callback on MouseDown
```

XuiPressButton grids display a text string.

Selecting a **XuiPressBox** grid generates a **Selection** callback message.

```
'
'
' ##########################
' #####  PressButton  #####
' ##########################
'
FUNCTION  PressButton ()
  STATIC  count
  STATIC  label
  $XuiPressButton = 4
'
  GetDisplayGrid ( @label )
'
' create window with XuiPressButton grid
'
  XuiCreateWindow      (@grid, @"XuiPressButton", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1,-1, $XuiPressButton, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"PressButton")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiPressButton")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  INC count
  text$ = ""
  text$ = text$ + "XuiPressButton\n\n"
  text$ = text$ + " has been selected\n\n   "
  text$ = text$ + STRING$(count) + " times"
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiPushButton




```
Kid 0 - XuiPushButton
  "Selection" callback on Enter KeyDown
  "Selection" callback on MouseDown + MouseUp
```

XuiPushButton grids display a text string.

Selecting a **XuiPushButton** grid generates a **Selection** callback message.

```
'
'
' ########################
' #####  PushButton  #####
' ########################
'
FUNCTION  PushButton ()
  STATIC  count
  STATIC  label
  $XuiPushButton = 5
'
  GetDisplayGrid ( @label )
'
' create window with XuiPushButton grid
'
  XuiCreateWindow      (@grid, @"XuiPushButton", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiPushButton, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"PushButton")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiPushButton")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  INC count
  text$ = ""
  text$ = text$ + "XuiPushButton\n\n  "
  text$ = text$ + "has been selected\n\n    "
  text$ = text$ + STRING$(count) + " times"
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiToggleButton


```
Kid 0 - XuiToggleButton
  "Selection" callback on MouseDown
     v0 : $$FALSE = not selected
          $$TRUE = selected
```

XuiToggleButton grids display a text string and toggle between selected and not selected.

Selecting a **XuiToggleButton** grid toggles it to the opposite state.

```
'
'
' ##########################
' #####  ToggleButton  #####
' ##########################
'
FUNCTION  ToggleButton ()
  STATIC  label
  $XuiToggleButton = 6
'
  GetDisplayGrid ( @label )
'
' create window with XuiToggleButton grid
'
  XuiCreateWindow      (@grid, @"XuiToggleButton", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(),-1,-1, $XuiToggleButton, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"ToggleButton")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiToggleButton")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  IF v0 THEN selected$ = "selected" ELSE selected$ = "not selected"
  text$ = ""
  text$ = text$ + "XuiToggleButton\n\n  "
  text$ = text$ + selected$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiRadioButton


```
Kid 0 - XuiRadioBox
  "Selection" callback on Enter KeyDown
  "Selection" callback on MouseDown
     v0 : $$FALSE = not selected
          $$TRUE = selected
```

XuiRadioButton grids display a text string and is show as not selected or selected depending upon whether its border is raised or lowered.

Selecting a **XuiRadioButton** grid first deselects all other **XuiRadioButton** grids in the same window and group, then selects itself. Thus no more than one **XuiRadioButton** grid in a group is ever selected.

XuiRadioButton grids generate a **Selection** callback message with **v0=$$FALSE** when deselected and **v0=$$TRUE** when selected.

A window with a single **XuiRadioButton** is not particularly useful, since once selected, the only ay the user can deselect it is by selecting another **XuiRadioButton** grid in the window, none of which exist.

```
'
'
' ##########################
' #####  RadioButton  #####
' ##########################
'
FUNCTION  RadioButton ()
  $XuiRadioButton = 7
'
  GetDisplayGrid ( @label )
'
' create window with XuiRadioButton grid
'
  XuiCreateWindow      (@grid, @"XuiRadioButton", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(),-1,-1, $XuiRadioButton, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"RadioButton")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiRadioButton")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  IF v0 THEN selected$ = "selected" ELSE selected$ = "not selected"
  text$ = "XuiRadioButton\n\n  "
  text$ = text$ + selected$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiScrollBarH


```
Kid 0 - XuiScrollBarH
  "OneLess" callback from MouseDown on left button
  "MuchLess" callback from MouseDown to left of slider
  "Change" callback from MouseDrag of slider
  "MuchMore" callback from MouseDown to right of slider
  "OneMore" callback from MouseDown on right button
     v0 : lowest   - lowest possible value of slider = 0
     v1 : low      - low position slider = left end of slider
     v2 : high     - high position of slider = right end of slider
     v3 : highest  - highest possible value of slider
```

XuiScrollBarH grids display the size and position of a subrange. The whole range often represents the longest visible line of text in a **XuiTextArea** grid, while the subrange is the visible portion of that line.

For example, the **XuiScrollBarH** image shown above roughly indicates that the leftmost 1/4 of the longest line is not visible because it is to the left of the visible area, the center 1/2 of the longest line is visible, and the rightmost 1/4 of the longest line is not visible because it is to the right of the visible area.

Another way to think of it is this: The length of the trough represents the total length of the longest line and the slider in the trough represents the visible portion of that line.

- "message" - caused by this action
- **"OneLess"** - **MouseDown** on the left button
- **"MuchLess"** - **MouseDown** in the trough to the left of the slider
- **"Change"** - **MouseDrag** the slider left or right
- **"MuchMore"** - **MouseDown** in the trough to the right of the slider
- **"OneMore"** - **MouseDown** on the right button

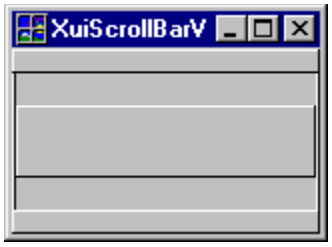
Dragging the slider left or right directly moves the slider in the trough. The other actions do not automatically move the slider - the code that receives the callback message can reposition the slider accordingly by sending the **XuiScrollBarH** grid a **"SetPosition"** message with arguments that specify the full range and subrange as follows:

```
v0 : lowest possible value of full range
v1 : low value of subrange
v2 : high value of subrange
v3 : highest possible value of full range
```

These values do not have to be in pixels, they can be in whatever integer units are appropriate to the application. For example, the full range might be zero to the number of characters on the longest line while the subrange is the character position of the leftmost and rightmost visible characters. The values returned in **v0,v1,v2,v3** by the callback messages are trough relative pixel locations of the trough and slider.

```
'
'
' ########################
' #####  ScrollBarH  #####
' ########################
'
FUNCTION  ScrollBarH ()
  $XuiScrollBarH = 8
'
  GetDisplayGrid ( @label )
'
' create window with XuiScrollBarH grid
'
  XuiCreateWindow      (@grid, @"XuiScrollBarH", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiScrollBarH, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"ScrollBarH")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiScrollBarH")
  XuiSendStringMessage ( grid, @"SetPosition", 0, 25, 75, 100, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "OneLess"     : GOSUB ScrollBarH
    CASE "MuchLess"    : GOSUB ScrollBarH
    CASE "Change"      : GOSUB ScrollBarH
    CASE "MuchMore"    : GOSUB ScrollBarH
    CASE "OneMore"     : GOSUB ScrollBarH
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' XuiScrollBarH messages
'
SUB ScrollBarH
  text$ = ""
  text$ = text$ + "XuiScrollBarH"
  text$ = text$ + "\n    grid = " + HEX$ (grid, 8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n  lowest = " + HEX$ (v0)
  text$ = text$ + "\n     low = " + HEX$ (v1)
  text$ = text$ + "\n    high = " + HEX$ (v2)
  text$ = text$ + "\n highest = " + HEX$ (v3)
  text$ = text$ + "\n     kid = " + HEX$ (kid)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiScrollBarV




```
Kid 0 - XuiScrollBarV
  "OneLess" callback from MouseDown on top button
  "MuchLess" callback from MouseDown above slider
  "Change" callback from MouseDrag of slider
  "MuchMore" callback from MouseDown below slider
  "OneMore" callback from MouseDown on bottom button
     v0 : lowest   - lowest possible value of slider = 0
     v1 : low      - low position slider = top end of slider
     v2 : high     - high position of slider = bottom end of slider
     v3 : highest  - highest possible value of slider
```

`XuiScrollBarV` grids display the size and position of a subrange. The whole range often represents the lines of text in an **`XuiTextArea`** grid, while the subrange represents the visible lines.

For example, the **`XuiScrollBarV`** image shown above roughly indicates that the first 1/4 of the lines are not visible because they are above the visible area, the center 1/2 of the lines are visible, and the last 1/4 of the lines are not visible because they are below the visible area.

Another way to think of it is this: The length of the trough represents the total number of lines of text and the slider in the trough represents the visible lines.

- "message" - caused by this action
- **`"OneLess"`** - **`MouseDown`** on the top button
- **`"MuchLess"`** - **`MouseDown`** in the trough above the slider
- **`"Change"`** - **`MouseDrag`** the slider up or down
- **`"MuchMore"`** - **`MouseDown`** in the trough below the slider
- **`"OneMore"`** - **`MouseDown`** on the bottom button

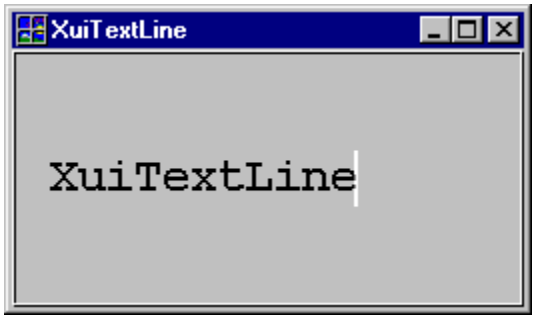
Dragging the slider up or down directly moves the slider in the trough. The other actions do not automatically move the slider - the code that receives the callback message can reposition the slider accordingly by sending the **`XuiScrollBarV`** grid a **`"SetPosition"`** message with arguments that specify the full range and subrange as follows:

```
v0 : lowest possible value of full range
v1 : low value of subrange
v2 : high value of subrange
v3 : highest possible value of full range
```

These values do not have to be in pixels, they can be in whatever integer units are appropriate to the application. For example, the full range might be zero to the number of lines while the subrange is the first and last visible lines. The values returned in **`v0,v1,v2,v3`** by the callback messages are trough relative pixel locations of the trough and slider.

```
'
'
' ########################
' #####  ScrollBarV  #####
' ########################
'
FUNCTION  ScrollBarV ()
  $XuiScrollBarV = 9
'
  GetDisplayGrid ( @label )
'
' create window with XuiScrollBarV grid
'
  XuiCreateWindow      (@grid, @"XuiScrollBarV", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiScrollBarV, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"ScrollBarV")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiScrollBarV")
  XuiSendStringMessage ( grid, @"SetPosition", 0, 25, 75, 100, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "OneLess"     : GOSUB ScrollBarV
    CASE "MuchLess"    : GOSUB ScrollBarV
    CASE "Change"      : GOSUB ScrollBarV
    CASE "MuchMore"    : GOSUB ScrollBarV
    CASE "OneMore"     : GOSUB ScrollBarV
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' XuiScrollBarV messages
'
SUB ScrollBarV
  text$ = ""
  text$ = text$ + "XuiScrollBarV"
  text$ = text$ + "\n    grid = " + HEX$ (grid, 8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n  lowest = " + HEX$ (v0,8)
  text$ = text$ + "\n     low = " + HEX$ (v1,8)
  text$ = text$ + "\n    high = " + HEX$ (v2,8)
  text$ = text$ + "\n highest = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiTextLine



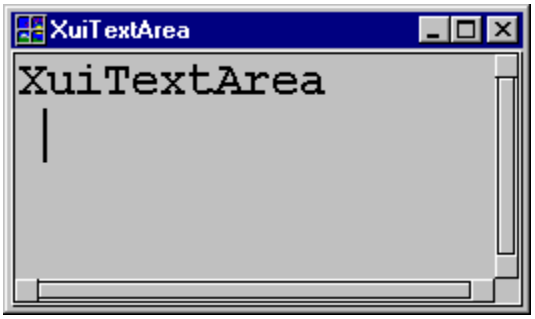

```
Kid 0 - XuiTextLine
  "TextEvent" callback from every KeyDown
  "Selection" callback from Enter KeyDown
```

XuiTextLine grids display a text string and accept user input text.

Pressing an Enter key while a **XuiTextLine** grid has keyboard focus generates a **Selection** callback message. Pressing an Enter key when the text cursor is anywhere in the line is equivalent - the Enter key is not inserted or appended to the text and the line is not broken into two parts.

```
'
'
' ######################
' #####  TextLine  #####
' ######################
'
FUNCTION  TextLine ()
  $XuiTextLine = 10
'
  GetDisplayGrid ( @label )
'
' create window with XuiTextLine grid
'
  XuiCreateWindow      (@grid, @"XuiTextLine", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiTextLine, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"TextLine")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiTextLine")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 0, @a$)
  text$ = ""
  text$ = text$ + "XuiTextLine"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + " : \"" + a$ + "\""
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiTextArea



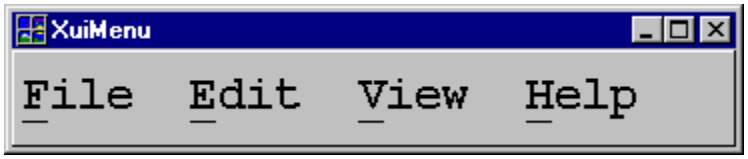

```
Kid 0 - XuiTextArea
  "TextEvent" callback from every KeyDown
  "Selection" callback from every KeyDown of Escape key
Kid 1 - XuiArea
Kid 2 - XuiScrollBarV
Kid 3 - XuiScrollBarH
```

XuiTextArea grids display a string array and accept user input text. Enter keystrokes at any character position move all following lines down a line.

Though text is displayed in kid #1, all normal messages should be sent to the **XuiTextArea** grid itself, as it manages the text and scroll bars.

```
'
'
' ######################
' #####  TextArea  #####
' ######################
'
FUNCTION  TextArea ()
  $XuiTextArea = 11
'
  GetDisplayGrid ( @label )
'
' create window with XuiTextArea grid
'
  XuiCreateWindow      (@grid, @"XuiTextArea", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiTextArea, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"TextArea")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiTextArea")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
END SUB
END FUNCTION
```

XuiMenu




```
Kid 0 - XuiMenu
  "Selection" callback from selecting a pulldown list item
     v0 : XuiMenuBar item - 1 is leftmost item
     v1 : XuiPullDown item - 0 is topmost item
```

XuiMenu grids display a list of catagories and let a user browse the items in each catagory by selecting the catagory name and select any item in any of the catagories presented.

When a catagory name like **File**, **Edit**, **View**, or **Help** is selected by a user action, a borderless window with an **XuiPullDown** grid drops down directly below the catagory name to display a number of items in that catagory. The user can select any item by dragging the mouse down to the item and releasing it, or by clicking on the catagory name to expose the **XuiPullDown** list, then on any item in the list.

XuiMenu grids create and manage the **XuiPullDown** windows without program intervention. When an item in an **XuiPullDown** list is selected, the **XuiMenu** grid generates a **Selection** message that reports the both catagory and the item in that catagory in **v0**, **v1**.

Catagories are numbered from 1 and items from 0. If a user selects the top item in the File catagory, the **Selection** callback message will contain **v0 = 1** and **v1 = 0**.

The catagory names and item names are both specified in the **TextArray** property of the **XuiMenu** grid, with item names indented below their left justified catagory names. For example, the following code segment sets up the **XuiMenu** example shown above.

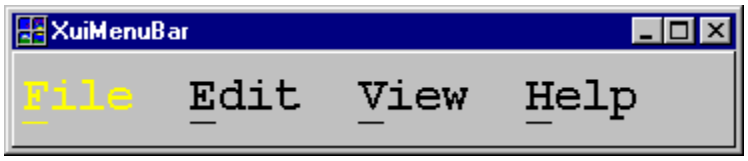
```
DIM menu$[15]              ' callback arguments
menu$[ 0] = "_File"
menu$[ 1] = " _Load"       ' v0 = 1 : v1 = 0
menu$[ 2] = " _Save"       ' v0 = 1 : v1 = 1
menu$[ 3] = " _Quit"       ' v0 = 1 : v1 = 2
menu$[ 4] = "_Edit"
menu$[ 5] = " _Cut"        ' v0 = 2 : v1 = 0
menu$[ 6] = " _Grab"       ' v0 = 2 : v1 = 1
menu$[ 7] = " _Paste"      ' v0 = 2 : v1 = 2
menu$[ 8] = "_View"
menu$[ 9] = " _First"      ' v0 = 3 : v1 = 0
menu$[10] = " _Last"       ' v0 = 3 : v1 = 1
menu$[11] = " _All"        ' v0 = 3 : v1 = 2
menu$[12] = "_Help"
menu$[13] = " _Contents"   ' v0 = 4 : v1 = 0
menu$[14] = " _Summary"    ' v0 = 4 : v1 = 1
menu$[15] = " _Index"      ' v0 = 4 : v1 = 2

XuiSendStringMessage (grid, @"SetTextArray", 0, 0, 0, 0, 0, @menu$[])
```

```
' ##################
' #####  Menu  #####
' ##################
'
FUNCTION  Menu ()
  $XuiMenu = 12
'
  GetDisplayGrid ( @label )
'
' create window with XuiMenu grid
'
  DIM menu$[15]             ' callback arguments
  menu$[ 0] = "_File"
  menu$[ 1] = " _Load"      ' v0 = 1 : v1 = 0
  menu$[ 2] = " _Save"      ' v0 = 1 : v1 = 1
  menu$[ 3] = " _Quit"      ' v0 = 1 : v1 = 2
  menu$[ 4] = "_Edit"
  menu$[ 5] = " _Cut"       ' v0 = 2 : v1 = 0
  menu$[ 6] = " _Grab"      ' v0 = 2 : v1 = 1
  menu$[ 7] = " _Paste"     ' v0 = 2 : v1 = 2
  menu$[ 8] = "_View"
  menu$[ 9] = " _First"     ' v0 = 3 : v1 = 0
  menu$[10] = " _Last"      ' v0 = 3 : v1 = 1
  menu$[11] = " _All"       ' v0 = 3 : v1 = 2
  menu$[12] = "_Help"
  menu$[13] = " _Contents"  ' v0 = 4 : v1 = 0
  menu$[14] = " _Summary"   ' v0 = 4 : v1 = 1
  menu$[15] = " _Index"     ' v0 = 4 : v1 = 2
'
  XuiCreateWindow      (@grid, @"XuiMenu", 100, 256, 256, 24, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiMenu, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Menu")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @menu$[])
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextArray", 0, 0, 0, 0, 0, @text$[])
  IF text$[] THEN
    u = UBOUND (text$[])
    menubar = 0
    FOR i = 0 TO u
      text$ = text$[i]
      IF text$ THEN
        char = text${0}
        IF ((char != ' ') AND (char != '   ')) THEN    ' not tab or space
          INC menubar
          IF (menubar = v0) THEN header = i : EXIT FOR
        END IF
      END IF
    NEXT i
    IF (v0 = menubar) THEN
      menu$ = text$[header]
      list = header + v1 + 1
      IF (list <= u) THEN list$ = text$[list]
      v0$ = " : Menu entry : \"" + menu$ + "\""
      v1$ = " : List entry : \"" + list$ + "\""
    END IF
  END IF
  text$ = "XuiMenu"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8) + v1$
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
```

```
END SUB
END FUNCTION
```

XuiMenuBar




```
Kid 0 - XuiMenuBar
  "Selection" callback from MouseDown or MouseDrag on item
     v0 : item - 1 is leftmost item
```

`XuiMenuBar` grids display a horizontal list of catagory names and let a user select any catagory.

When a catagory name like **`File`**, **`Edit`**, **`View`**, or **`Help`** is selected by a user action, an **`XuiMenuBar`** grid generates a callback message with the selected catagory in **`v0`**.

Catagories are numbered from 1. If the File catagory is selected, the **`Selection`** callback message will contain **`v0 = 1`**.

The catagory names are the **`TextString`** property, with at least one space between each catagory name.

```
'
'
' ######################
' #####  MenuBar  #####
' ######################
'
FUNCTION  MenuBar ()
  $XuiMenuBar = 13
'
  GetDisplayGrid ( @label )
'
' create window with XuiMenuBar grid
'
  XuiCreateWindow      (@grid, @"XuiMenuBar", 100, 256, 256, 24, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiMenuBar, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"MenuBar")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"_File  _Edit  _View  _Help")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 0, @text$)
  IFZ text$ THEN EXIT SUB
  item = $$FALSE
  done = $$FALSE
  pos = $$FALSE
  DO UNTIL (item >= v0)
    INC item
    item$ = XstNextField$ (@text$, @pos, @done)
  LOOP UNTIL done
  IF item$ THEN
    IF (v0 = item) THEN v0$ = " : item : \"" + item$ + "\""
  END IF
'
  text$ = "XuiMenuBar"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiPullDown


```
Kid 0 - XuiPullDown
  "TextEvent" callback from any KeyDown
  "Selection" callback from MouseUp on item
     v0 : selected item
  "Selection" callback from Enter KeyDown
     v0 : selected item
  "Selection" callback from hotkey KeyDown
     v0 : selected item (key = underlined key)
```

XuiPullDown grids display a list of items and let a user select any item by selecting the item name.

When an item name like **Load, Save, Rename, Delete, Quit** is selected by a user action, an **XuiPullDown** grid generates a callback message with the selected item in **v0**.

Items are numbered from 0 at the top. If the **Load** item is selected, for example, the **Selection** callback message will contain **v0 = 0**.

The item names are the **TextArray** property.

An item in an **XuiPullDown** grid is selected by **MouseUp** when the mouse cursor is over an item name, by **KeyDown** of the Enter key when an item is highlighted, and by **KeyDown** of a key corresponding to an underlined character in an item in the **XuiPullDown** list.

The highlighted item can be changed with **UpArrow** and **DownArrow**.

```
'
'
' ######################
' #####  PullDown  #####
' ######################
'
FUNCTION  PullDown ()
  $XuiPullDown = 14
'
  GetDisplayGrid ( @label )
'
' create window with XuiPullDown grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiPullDown", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiPullDown, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"PullDown")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @short$[])
  XuiSendStringMessage ( grid, @"Resize", 0, 0, 200, 80, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextArrayLine", v0, 0, 0, 0, 0, @text$)
  IF text$ THEN v0$ = " : item : " + text$
'
  text$ = "XuiPullDown"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiList


```
Kid 0 - XuiList
  "TextEvent" callback from any KeyDown
  "Selection" callback from double click on an item
     v0 : selected item
  "Selection" callback from Enter KeyDown
     v0 : selected item
```

XuiList grids display a list of items and let a user select any item by selecting the item name.

When an item name like **Zero, One, Two, ... Five, Six, Seven** is selected by a user action, an **XuiList** grid generates a callback message with the selected item in **v0**.

Items are numbered from 0 at the top. If the **Zero** item is selected, for example, the **Selection** callback message will contain **v0 = 0**.

The item names are the **TextArray** property.

An item in an **XuiList** grid is selected by double clicking an item, and by **KeyDown** of the **Enter** key when an item is highlighted.

The highlighted item can be changed with **UpArrow** and **DownArrow**.

```
'
'
' ##################
' #####  List  #####
' ##################
'
FUNCTION  List ()
  $XuiList = 15
'
  GetDisplayGrid ( @label )
'
' create window with XuiList grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiList", 100, 256, 200, 200, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiList, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"List")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @short$[])
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextArrayLine", v0, 0, 0, 0, 0, @text$)
  IF text$ THEN v0$ = " : item : " + text$
'
  text$ = "XuiList"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiMessage1B


```
Kid 0 - XuiMessage1B
Kid 1 - XuiLabel
Kid 2 - XuiPushButton
  "Selection" callback from MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
```

XuiMessage1B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user selects its **XuiPushButton** grid.

```
'
'
' ######################
' #####  Message1B  #####
' ######################
'
FUNCTION  Message1B ()
  $XuiMessage1B = 16
'
  GetDisplayGrid ( @label )
'
' create window with XuiMessage1B grid
'
  XuiCreateWindow      (@grid, @"XuiMessage1B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiMessage1B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Message1B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiMessage1B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  text$ = "XuiMessage1B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiMessage2B


```
Kid 0 - XuiMessage2B
Kid 1 - XuiLabel
Kid 2 - XuiPushButton
  "Selection" callback from MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
Kid 3 - XuiPushButton
  "Selection" callback from MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
```

XuiMessage2B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user selects either of the **XuiPushButton** grids.

```
'
'
' ######################
' #####  Message2B  #####
' ######################
'
FUNCTION  Message2B ()
  $XuiMessage2B = 17
'
  GetDisplayGrid ( @label )
'
' create window with XuiMessage2B grid
'
  XuiCreateWindow      (@grid, @"XuiMessage2B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiMessage2B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Message2B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiMessage2B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  text$ = "XuiMessage2B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiMessage3B


```
Kid 0 - XuiMessage3B
Kid 1 - XuiLabel
Kid 2 - XuiPushButton
  "Selection" callback from MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
Kid 3 - XuiPushButton
  "Selection" callback from MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
Kid 4 - XuiPushButton
  "Selection" callback from MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
```

XuiMessage3B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user selects any of the **XuiPushButton** grids.

```
'
'
' ########################
' #####  Message3B  #####
' ########################
'
FUNCTION  Message3B ()
  $XuiMessage3B = 18
'
  GetDisplayGrid ( @label )
'
' create window with XuiMessage3B grid
'
  XuiCreateWindow      (@grid, @"XuiMessage3B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiMessage3B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Message3B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiMessage3B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  text$ = "XuiMessage3B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

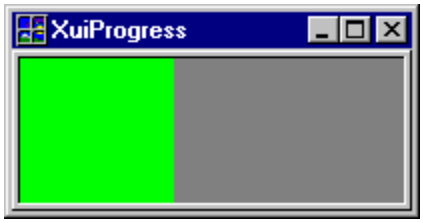
XuiMessage4B


```
Kid 0 - XuiMessage4B
Kid 1 - XuiLabel
Kid 2 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
Kid 3 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
Kid 4 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
Kid 5 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback from Enter KeyDown when button has focus
```

XuiMessage4B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user selects any of the **XuiPushButton** grids.

```
'
'
' ######################
' #####  Message4B  #####
' ######################
'
FUNCTION  Message4B ()
  $XuiMessage4B = 19
'
  GetDisplayGrid ( @label )
'
' create window with XuiMessage4B grid
'
  XuiCreateWindow      (@grid, @"XuiMessage4B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiMessage4B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Message4B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiMessage4B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  text$ = "XuiMessage4B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiProgress




```
Kid 0 - XuiProgress - does not initiate callbacks

  "SetValues" message
     v0 = position
     v1 = full scale
```

XuiProgress grids display a band of color on a backgroud to represent percentage of completion of something analogous.

"GetValues" returns the position and full scale in **v0,v1**.
"SetValues" sets the position and full scale with **v0,v1**.

```
'
'
' ######################
' #####  Progress  #####
' ######################
'
FUNCTION  Progress ()
  $XuiProgress = 20
'
  delta = 2
  GetDisplayGrid ( @label )
'
' create window with XuiProgress grid
'
  XuiCreateWindow      (@grid, @"XuiProgress", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiProgress, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Progress")
  XuiSendStringMessage ( grid, @"SetValues", 50, 100, 0, 0, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
                         XuiSendStringMessage (grid, @"GetValues", @v0, @v1, 0, 0, 0, 0)
                         v0 = v0 + delta
                         IF (v0 > 100) THEN v0 = v0 - delta - delta : delta = -delta
                         IF (v0 < 0) THEN v0 = v0 - delta - delta : delta = -delta
                         XuiSendStringMessage (grid, @"SetValues", v0, v1, 0, 0, 0, 0)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  text$ = "XuiProgress"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiDialog2B


```
Kid 0 - XuiDialog2B
Kid 1 - XuiLabel
Kid 2 - XuiTextLine
  "TextEvent" callback on any KeyDown
  "Selection" callback on Enter KeyDown
Kid 3 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
Kid 4 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
```

XuiDialog2B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user presses the **Enter** key when the **XuiTextLine** has keyboard focus, or when the user selects either of the **XuiPushButton** grids.

```
'
'
' ######################
' #####  Dialog2B  #####
' ######################
'
FUNCTION  Dialog2B ()
  $XuiDialog2B = 17
'
  GetDisplayGrid ( @label )
'
' create window with XuiDialog2B grid
'
  XuiCreateWindow      (@grid, @"XuiDialog2B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiDialog2B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Dialog2B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiDialog2B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 2, @a$)
  text$ = "XuiDialog2B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + " : " + a$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiDialog3B


```
Kid 0 - XuiDialog3B
Kid 1 - XuiLabel
Kid 2 - XuiTextLine
  "TextEvent" callback on any KeyDown
  "Selection" callback on Enter KeyDown
Kid 3 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
Kid 4 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
Kid 5 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
```

XuiDialog3B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user presses the **Enter** key when the **XuiTextLine** has keyboard focus, or when the user selects any of the **XuiPushButton** grids.

```
'
'
' ######################
' #####  Dialog3B  #####
' ######################
'
FUNCTION  Dialog3B ()
  $XuiDialog3B = 18
'
  GetDisplayGrid ( @label )
'
' create window with XuiDialog3B grid
'
  XuiCreateWindow      (@grid, @"XuiDialog3B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiDialog3B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Dialog3B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiDialog3B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 2, @a$)
  text$ = "XuiDialog3B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + " : " + a$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiDialog4B


```
Kid 0 - XuiDialog4B
Kid 1 - XuiLabel
Kid 2 - XuiTextLine
  "TextEvent" callback on any KeyDown
  "Selection" callback on Enter KeyDown
Kid 3 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
Kid 4 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
Kid 5 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
Kid 6 - XuiPushButton
  "Selection" callback on MouseDown + MouseUp
  "Selection" callback on Enter KeyDown when button has focus
```

XuiDialog4B grids display a text string on the **XuiLabel** grid and generate **Selection** callback messages when the user presses the **Enter** key when the **XuiTextLine** has keyboard focus, or when the user selects any of the **XuiPushButton** grids.

```
'
'
' ######################
' #####  Dialog4B  #####
' ######################
'
FUNCTION  Dialog4B ()
  $XuiDialog4B = 19
'
  GetDisplayGrid ( @label )
'
' create window with XuiDialog4B grid
'
  XuiCreateWindow      (@grid, @"XuiDialog4B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiDialog4B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Dialog4B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiDialog4B")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 2, @a$)
  text$ = "XuiDialog4B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + " : " + a$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiDropButton

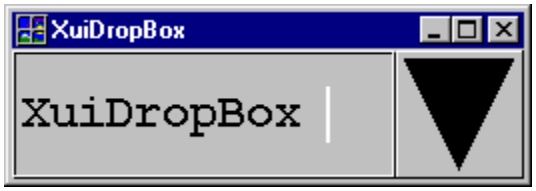

```
Kid 0 - XuiDropButton
  "Selection" callback when XuiPullDown item selected
     v0 : item - 0 is topmost item
```

XuiDropButton grids display a text string on the **XuiDropButton** grid, and a list of items in an **XuiPullDown** that toggles up and down when the **XuiDropButton** is selected. **XuiDropButton** grids generate **Selection** callback messages when the user selects any item from the **XuiPullDown** grid.

The **XuiDropButton** grid manages the **XuiPullDown** window, so the text on the **XuiDropButton** is its **TextString** property and the **XuiPullDown** list is the its **TextArray** property.

```
'
'
' ########################
' #####  DropButton  #####
' ########################
'
FUNCTION  DropButton ()
  $XuiDropButton = 24
'
  GetDisplayGrid ( @label )
'
' create window with XuiDropButton grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiDropButton", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiDropButton, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"DropButton")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiDropButton")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @short$[])
  XuiSendStringMessage ( grid, @"Resize", 0, 0, 200, 80, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextArrayLine", v0, 0, 0, 0, 0, @text$)
  IF text$ THEN v0$ = " : item : " + text$
'
  text$ = "XuiDropButton"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiDropBox




```
Kid 0 - XuiDropBox
  "Selection" callback when XuiPullDown item selected
     v0 : item - 0 is topmost item
Kid 1 - XuiTextLine
  "Selection" callback from Enter KeyDown when grid has focus
     v0 : item if text string is also in XuiPullDown list, else -1
Kid 2 - XuiPressButton
```

XuiDropBox grids display a text string in the **XuiTextLine** grid, and a list of items in an **XuiPullDown** that toggles up and down when the **XuiDropBox** button is selected. **XuiDropBox** grids generate **Selection** callback messages when the user presses an **Enter** key when the **XuiTextLine** has keyboard focus, and when the user selects any item in the **XuiPullDown** grid.

The **XuiDropBox** grid manages the **XuiPullDown** window, so the **XuiPullDown** list is the its **TextArray** property of the **XuiDropBox**.

```
'
'
' ######################
' #####  DropBox  #####
' ######################
'
FUNCTION  DropBox ()
  $XuiDropBox = 25
'
  GetDisplayGrid ( @label )
'
' create window with XuiDropBox grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiDropBox", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiDropBox, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"DropBox")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiDropBox")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @short$[])
  XuiSendStringMessage ( grid, @"Resize", 0, 0, 200, 80, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 1, @text$)
  IF text$ THEN v0$ = " : item : \"" + text$ + "\""
'
  text$ = "XuiDropBox"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiListButton


```
Kid 0 - XuiListButton
  "Selection" callback when XuiList item selected
     v0 : item - 0 is topmost item
```

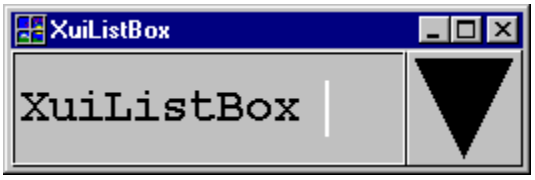
XuiListButton grids display a text string on the **XuiListButton** grid, and a list of items in an **XuiList** grid window that toggles up and down when the **XuiListButton** is selected.

XuiListButton grids generate **Selection** callback messages when the user selects any item from the **XuiList** grid.

The **XuiListButton** grid manages the **XuiList** window, so the text on the **XuiListButton** is its **TextString** property and the **XuiList** items are the **TextArray** property of the **XuiListButton**.

```
'
'
' ########################
' #####  ListButton  #####
' ########################
'
FUNCTION  ListButton ()
  $XuiListButton = 26
'
  GetDisplayGrid ( @label )
'
' create window with XuiListButton grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiListButton", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiListButton, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"ListButton")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiListButton")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @short$[])
  XuiSendStringMessage ( grid, @"Resize", 0, 0, 200, 80, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextArrayLine", v0, 0, 0, 0, 0, @text$)
  IF text$ THEN v0$ = " : item : " + text$
'
  text$ = "XuiListButton"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiListBox




```
Kid 0 - XuiListButton
  "Selection" callback when XuiList item selected
     v0 : item - 0 is topmost item
Kid 1 - XuiTextLine
  "Selection" callback from Enter KeyDown when grid has focus
     v0 : item if text string is also in XuiPullDown list, else -1
Kid 2 - XuiPressButton
```

XuiListBox grids display a text string in the **XuiTextLine** grid, and a list of items in an **XuiList** grid window that toggles up and down when the **XuiListBox** button is selected.

XuiListBox grids generate **Selection** callback messages when the user presses an **Enter** key when the **XuiTextLine** has keyboard focus, and when the user selects any item in the **XuiList** grid.

The **XuiDropBox** grid manages the **XuiList** window, so the **XuiList** items are the **TextArray** property of the **XuiListBox**.

```
'
'
' ######################
' #####  ListBox  #####
' ######################
'
FUNCTION  ListBox ()
  $XuiListBox = 27
'
  GetDisplayGrid ( @label )
'
' create window with XuiListBox grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiListBox", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiListBox, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"ListBox")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 0, @"XuiListBox")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 0, @short$[])
  XuiSendStringMessage ( grid, @"Resize", 0, 0, 200, 80, 0, 0)
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 1, @text$)
  IF text$ THEN v0$ = " : item : \"" + text$ + "\""
'
  text$ = "XuiListBox"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + v0$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiRange


```
Kid 0 - XuiRange
Kid 1 - XuiLabel
Kid 2 - XuiPressButton
Kid 3 - XuiPressButton
  "Selection" callback on MouseDown on top or bottom button
     v0 : current value
     v1 : delta = + value for up : -value for down
     v2 : minimum value
     v3 : maximum value
```

XuiRange grids display a numeric value that can be slewed up or down with the up and down buttons. **XuiRange** grids operate with four values that can be read or set with **GetValues** and **SetValues** as follows:

```
XuiSendStringMessage (grid, "GetValues", @v0, @v1, @v2, @v3, 0, 0)
XuiSendStringMessage (grid, "SetValues", v0, v1, v2, v3, 0, 0)
```

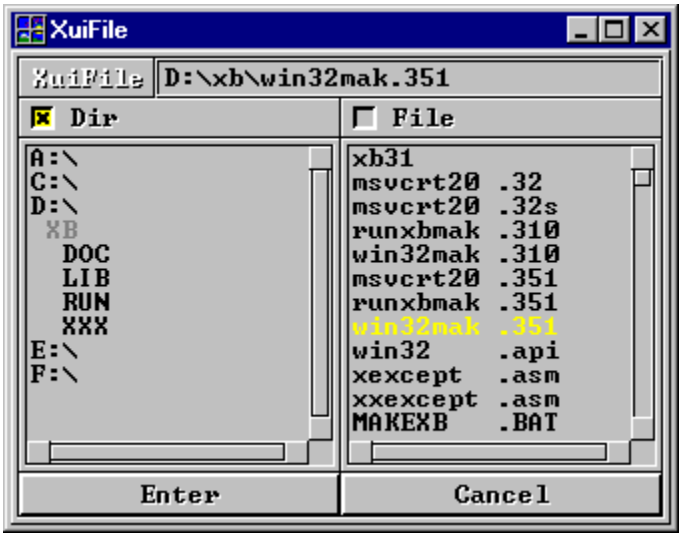
v0 : current value : displayed on the **XuiRange** grid
v1 : delta : amount to add or subtract when up or down button selected
v2 : minimum value : current value will not decrease beyond this value
v3 : maximum value : current value will not increase beyond this value

The default values of **v0,v1,v2,v3** are **0, 1, 0, 100**.

XuiRange grids generate **Selection** callback messages when the user selects the up or down buttons.

```
'
'
' ###################
' #####  Range  #####
' ###################
'
FUNCTION  Range ()
  STATIC  label
  $XuiRange = 28
'
  GetDisplayGrid ( @label )
'
' create window with XuiRange grid
'
  XuiCreateWindow      (@grid, @"XuiRange", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiRange, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Range")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiRange")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  text$ = "XuiRange\n"
  text$ = text$ + "    grid = " + HEX$ (grid, 8) + "\n"
  text$ = text$ + " message = " + message$ + "\n"
  text$ = text$ + "      v0 = " + HEX$ (v0,8) + " : value\n"
  text$ = text$ + "      v1 = " + HEX$ (v1,8) + " : added\n"
  text$ = text$ + "      v2 = " + HEX$ (v2,8) + " : minimum\n"
  text$ = text$ + "      v3 = " + HEX$ (v3,8) + " : maximum\n"
  text$ = text$ + "     kid = " + HEX$ (kid,8) + "\n"
  text$ = text$ + "     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiFile



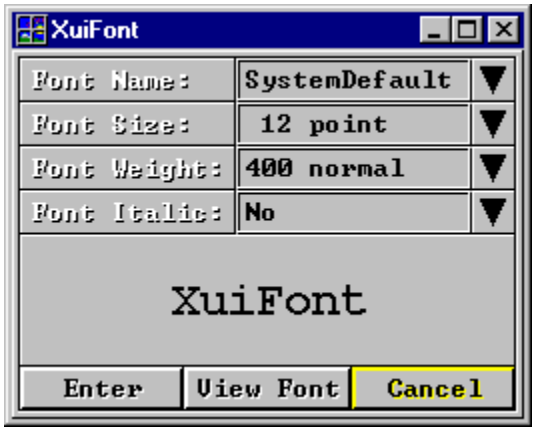

```
Kid 0 - XuiFile
Kid 1 - XuiLabel
Kid 2 - XuiTextLine
  "TextEvent" callback from any KeyDown
  "Selection" callback from Enter KeyDown
Kid 3 - XuiCheckBox
Kid 4 - XuiCheckBox
Kid 5 - XuiList
Kid 6 - XuiList
  "Selection" callback from file selection
Kid 7 - XuiPushButton
  "Selection" callback from button selection
Kid 8 - XuiPushButton
  "Selection" callback from button selection
```

XuiFile grids display a list of directories, a list of files in the selected directory, and a filename. **XuiFile** grids let the user navigate through filesystems by selecting directories and files.

XuiFile grids generate **Selection** callback messages when the user selects a specific file item in the **XuiList** of files, and when the user enters a path and filename in the **XuiTextLine**.

```
'
'
' ##################
' #####  File  #####
' ##################
'
FUNCTION  File ()
  STATIC  label
  $XuiFile = 29
'
  GetDisplayGrid ( @label )
'
' create window with XuiFile grid
'
  XuiCreateWindow      (@grid, @"XuiFile", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiFile, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"File")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiFile")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  IF (v0 < 0) THEN
    file$ = "cancel"
  ELSE
    xx$ = ""
    attributes = 0
    XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 2, @file$)
    IF file$ THEN XstGetFileAttributes (@file$, @attributes)
    IFZ attributes THEN
      xx$ = xx$ + " $$FileNonexistent"
    ELSE
      SELECT CASE ALL TRUE
        CASE (attributes AND $$FileReadOnly)    : xx$ = xx$ + " $$FileReadOnly"
        CASE (attributes AND $$FileHidden)      : xx$ = xx$ + " $$FileHidden"
        CASE (attributes AND $$FileSystem)      : xx$ = xx$ + " $$FileSystem"
        CASE (attributes AND $$FileDirectory)   : xx$ = xx$ + " $$FileDirectory"
        CASE (attributes AND $$FileArchive)     : xx$ = xx$ + " $$FileArchive"
        CASE (attributes AND $$FileNormal)      : xx$ = xx$ + " $$FileNormal"
        CASE (attributes AND $$FileTemporary)   : xx$ = xx$ + " $$FileTemporary"
        CASE (attributes AND $$FileAtomicWrite) : xx$ = xx$ + " $$FileAtomicWrite"
        CASE (attributes AND $$FileExecutable)  : xx$ = xx$ + " $$FileExecutable"
      END SELECT
    END IF
  END IF
'
  text$ = "XuiFile\n"
  text$ = text$ + "    grid = " + HEX$ (grid, 8) + "\n"
  text$ = text$ + " message = " + message$ + "\n"
  text$ = text$ + "      v0 = " + HEX$ (v0,8) + "   : " + file$ + "\n"
  text$ = text$ + "      v1 = " + HEX$ (v1,8) + "   :" + xx$ + "\n"
  text$ = text$ + "      v2 = " + HEX$ (v2,8) + "\n"
  text$ = text$ + "      v3 = " + HEX$ (v3,8) + "\n"
  text$ = text$ + "     kid = " + HEX$ (kid,8) + "\n"
  text$ = text$ + "     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiFont




```
Kid  0 - XuiFont
Kid  1 - XuiLabel
Kid  2 - XuiListBox
Kid  3 - XuiLabel
Kid  4 - XuiListBox
Kid  5 - XuiLabel
Kid  6 - XuiListBox
Kid  7 - XuiLabel
Kid  8 - XuiListBox
Kid  9 - XuiLabel
Kid 10 - XuiPushButton
  "Selection" callback from button selection
     v0 : font number
Kid 11 - XuiPushButton
Kid 12 - XuiPushButton
  "Selection" callback from button selection
     v0 : -1 means cancel
```

XuiFont grids let users specify, view, and select a font on the basis of font typeface name, size, weight, and italic properties.

XuiFont grids generate **Selection** callback messages when the user selects the **Enter** or **Cancel** button. The center button displays a sample of the font in the **XuiLabel** grid but generates no callback.

```
'
'
' ##################
' #####  Font  #####
' ##################
'
FUNCTION  Font ()
  STATIC  label
  $XuiFont = 30
'
  GetDisplayGrid ( @label )
'
' create window with XuiFont grid
'
  XuiCreateWindow      (@grid, @"XuiFont", 100, 256, 256, 128, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(), -1, -1, $XuiFont, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"Font")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiFont")
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  v0$ = ""
  v1$ = ""
  v2$ = ""
  v3$ = ""
  IF (v0 < 0) THEN
    v0$ = " : cancel"
  ELSE
    XgrGetFontInfo (v0, @font$, @size, @weight, @italic, @angle)
    v0$ =  " : font #   typeface : \"" + font$ + "\""
    v1$ =  "           20 * size : " + RJUST$(STRING$(size),4)
    v2$ =  "              weight : " + RJUST$(STRING$(weight),4)
    v3$ =  "              italic : " + RJUST$(STRING$(italic),4)
  END IF
'
  text$ = "XuiFont\n"
  text$ = text$ + "    grid = " + HEX$ (grid, 8) + "\n"
  text$ = text$ + " message = " + message$ + "\n"
  text$ = text$ + "      v0 = " + HEX$ (v0,8) + v0$ + "\n"
  text$ = text$ + "      v1 = " + HEX$ (v1,8) + v1$ + "\n"
  text$ = text$ + "      v2 = " + HEX$ (v2,8) + v2$ + "\n"
  text$ = text$ + "      v3 = " + HEX$ (v3,8) + v3$ + "\n"
  text$ = text$ + "     kid = " + HEX$ (kid,8) + "\n"
  text$ = text$ + "     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```

XuiListDialog2B


```
Kid 0 - XuiListDialog2B
Kid 1 - XuiLabel
Kid 2 - XuiList
  "Selection" callback from item selected
Kid 3 - XuiTextLine
  "TextEvent" callback from any KeyDown
  "Selection" callback from Enter KeyDown
Kid 4 - XuiPushButton
  "Selection" callback from button selection
Kid 5 - XuiPushButton
  "Selection" callback from button selection
```

XuiListDialog2B grids display an **XuiList** of items and an **XuiTextLine** text entry grid.

XuiListDialog2B grids generate **Selection** callback messages when the user selects an item in the **XuiList** grid, when the user presses the **Enter** key when the **XuiTextLine** grid has keyboard focus, and when the user selects the **Enter** or **Cancel** button.

```
'
'
' ##########################
' #####  ListDialog2B  #####
' ##########################
'
FUNCTION  ListDialog2B ()
  $XuiListDialog2B = 31
'
  GetDisplayGrid ( @label )
'
' create window with XuiListDialog2B grid
'
  DIM short$[7]
  short$[0] = "Zero"
  short$[1] = "One"
  short$[2] = "Two"
  short$[3] = "Three"
  short$[4] = "Four"
  short$[5] = "Five"
  short$[6] = "Six"
  short$[7] = "Seven"
'
  XuiCreateWindow      (@grid, @"XuiListDialog2B", 100, 256, 200, 80, 0, "")
  XuiSendStringMessage ( grid, @"SetCallback", grid, &XuiQueueCallbacks(),-1,-1, $XuiListDialog2B, -1)
  XuiSendStringMessage ( grid, @"SetGridName", 0, 0, 0, 0, 0, @"ListDialog2B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 1, @"XuiListDialog2B")
  XuiSendStringMessage ( grid, @"SetTextString", 0, 0, 0, 0, 3, @"TextLine")
  XuiSendStringMessage ( grid, @"SetTextArray", 0, 0, 0, 0, 2, @short$[])
  XuiSendStringMessage ( grid, @"DisplayWindow", 0, 0, 0, 0, 0, 0)
'
' convenience function message loop
'
  DO
    XgrProcessMessages (1)
    DO WHILE XuiGetNextCallback (@grid, @message$, @v0, @v1, @v2, @v3, @kid, @r1$)
      GOSUB Callback
    LOOP
  LOOP
  RETURN
'
' callback
'
SUB Callback
  win = kid >> 16
  kid = kid AND 0xFF
  SELECT CASE message$
    CASE "CloseWindow" : QUIT (0)
    CASE "Selection"   : GOSUB Selection
    CASE ELSE          : DisplayCallback (grid, @message$, v0, v1, v2, v3, kid, @r1$)
  END SELECT
END SUB
'
' Selection message
'
SUB Selection
  SELECT CASE kid
    CASE 2  : XuiSendStringMessage (grid, @"GetTextArrayLine", v0, 0, 0, 0, 2, @a$)
    CASE 3  : XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 3, @a$)
    CASE 4  : XuiSendStringMessage (grid, @"GetTextString", 0, 0, 0, 0, 3, @a$)
    CASE 5  : a$ = "cancel"
  END SELECT
'
  text$ = "XuiListDialog2B"
  text$ = text$ + "\n    grid = " + HEX$ (grid,8)
  text$ = text$ + "\n message = " + message$
  text$ = text$ + "\n   state = " + HEX$ (v0,8) + " : " + a$
  text$ = text$ + "\n      v1 = " + HEX$ (v1,8)
  text$ = text$ + "\n      v2 = " + HEX$ (v2,8)
  text$ = text$ + "\n      v3 = " + HEX$ (v3,8)
  text$ = text$ + "\n     kid = " + HEX$ (kid,8)
  text$ = text$ + "\n     r1$ = " + r1$
  XuiSendStringMessage (label, @"SetTextString", 0, 0, 0, 0, 0, @text$)    ' set message text
  XuiSendStringMessage (label, @"Redraw", 0, 0, 0, 0, 0, 0)                ' redraw label
END SUB
END FUNCTION
```