FICO™ Xpress Optimization Suite

Xpress Application Developer

Reference manual

Release 1.2.1

Last update 02 June, 2009

Published by Fair Isaac Corporation

How to Contact the Xpress Team

Information, Sales and Licensing

USA, CANADA AND ALL AMERICAS

Email: XpressSalesUS@fico.com

WORLDWIDE

Email: XpressSalesUK@fico.com

Tel: +44 1926 315862
Fax: +44 1926 315854

FICO, Xpress team
Leam House, 64 Trinity Street
Leamington Spa
Warwickshire CV32 5YN
UK

Product Support

Email: Support@fico.com
(Please include *'Xpress'* in the subject line)

Telephone:

NORTH AMERICA
Tel (toll free): +1 (877) 4FI-SUPP
Fax: +1 (402) 496-2224

EUROPE, MIDDLE EAST, AFRICA
Tel: +44 (0) 870-420-3777
UK (toll free): 0800-0152-153
South Africa (toll free): 0800-996-153
Fax: +44 (0) 870-420-3778

ASIA-PACIFIC, LATIN AMERICA, CARIBBEAN
Tel: +1 (415) 446-6185
Brazil (toll free): 0800-891-6146

For the latest news and Xpress software and documentation updates, please visit the Xpress website at http://www.fico.com/xpress or subscribe to our mailing list.

Contents

Chapter 1

XAD Applications

High level applications of Mosel and XAD

- Build an **interactive personnel assignment** optimization application.
- Visualize the performance in time of an **asset portfolio model**.
- Prototype a **bin packing** model.
- **Cutting stock** problems too difficult to understand? Use XAD to clear up the picture.
- Deploy a **facility location** optimization application, with **interactive GIS** functionality.
- Prototype and deploy a **vehicle routing** application.
- Visualize **strategic capacity planning** with a 30-year horizon.
- Take advantage of **parallel computing** and visualize concurrent optimization runs.
- Write a simple **text editor in 20 lines or less** of Mosel code.
- Build breakthrough **journaling/monitoring features** which record how users interact with your application.
- Quickly build a **visualization platform** for your data, results, or both.
- Build data **input forms** for your users.
- Easily load and implement an application GUI created in the IVE resource editor.

Development with Mosel and XAD is contiguous and seamless. Construct the user interface using the IVE drag and drop Resource Editor (see section 11.1 for an example), or write Mosel code to build both the user interface as well as the mathematical optimization model, data input/output and pre/postprocessing. Save an order of magnitude of development time by not having to switch editors, compilers, data paths, or even developers. The Mosel environment combined with XAD enables fast and intuitive development.

Chapter 2

GUI creation with XAD

Xpress Application Developer (XAD) is an extension of the Xpress-Mosel modeling and programming language. Xpress Application Developer extends the functionality of Mosel with a set of functions and procedures for creating standard user interfaces. As a result, Mosel can be used as a modeling and programming language for *complete optimization application development*, from the mathematical representation of a problem to developing the user interface.

Xpress Application Developer can save significant amounts of time when experimenting with an optimization problem because the OR practitioner no longer needs to interface Mosel with VB, or C++, or Java in order to build a user application. Most features needed for GUI-based application development are now available through the powerful yet easy to use abstractions in the Mosel language.

An XAD user interface is composed of windows that can be opened or closed. Each window may contain any number of XAD objects such as lists, buttons, checkboxes, etc. Every window as well as every XAD object has a *unique* integer identifier (referred to as *id*) associated with it. The identifier is associated with the graphical object at creation and is henceforth used to refer to the respective object later in the program/model.

The user may construct the GUI for the windows using either the XAD Resource Editor (part of IVE and demonstrated in the 11.1 towards the end of this manual), or purely through Mosel code. When creating the GUI(s) using the resource editor the layout and basic information about the XAD objects is stored within an XML-like resource file and loaded directly in to XAD using `XADloadresource`. As a consequence of having loaded all of the XAD objects in the background you will not automatically have a reference to their *ids*. However, you will also not have had to write out repeated lines of code to layout the objects within the window. In order to make use of resource-loaded objects you must firstly retrieve their *id* using `XADgetid`; you may then use them as you would any XAD object. Resource-loaded objects also differ from their Mosel code generated counterparts in one other manner: they all have a name which may be used to both retrieve their *id*, and to call object-event specific *events* .

There are advantages and disadvantages to both the resource files and Mosel code approaches:

1. **Resource File:**

   (a) *Advantage*: Layout design is fast and easy using the IVE drag and drop resource editor.

   (b) *Advantage*: The window and all of its objects may be loaded in Mosel using one function call.

   (c) *Advantage*: Events callbacks for each event may be specific to each object and window. These are called simply by concatenation of the object's name, the window's name and the event name, i.e.: `Button_MainWindow_PRESSED`

(d) *Disadvantage*: To use any objects within the Mosel code you must retrieve the *id* using `XADgetid`.

2. **Pure Mosel Code:**

   (a) *Advantage*: You do not require the extra resource file(s) to setup the window layout.

   (b) *Advantage*: You can dynamically create the window layout at runtime.

   (c) *Disadvantage*: If you wish to use the new form of event callback then you need to assign the object(s) names. Otherwise you must handle all of the events through the function specified in `XADseteventcallback` procedure.

   (d) *Disadvantage*: If you want to change the layout of objects on a form you need to search through the code and compare positions and sizes numerically.

When the user interacts with an XAD object, *events* are generated. For example pressing a button generates an `XAD_EVENT_PRESSED`; selecting an item in a list generates an `XAD_EVENT_SELECTION` event. Events are processed in a callback routine written by the user. The behavior of the user interface is determined by how the application responds to events. Any Mosel code can be written for dealing with an event, including XAD statements for altering the state of the user interface. This allows great flexibility in dealing with Mosel data as well as interacting with the GUI.

The following, simple, example will demonstrate the principles of creating and managing an XAD user interface using pure Mosel code. For an example using resource files see the example in section 11.1 towards the end of this manual; for detailed usage guides to the various resource editor controls see the IVE help.

Suppose we want to write a Mosel program which displays a window asking for a number that is needed later in the model:

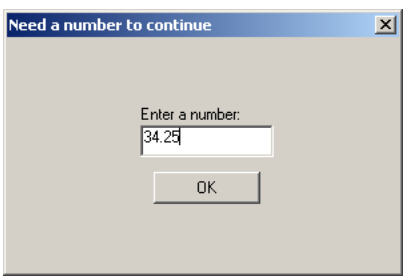


Figure 2.1: Simple input window example

Let's examine the code:

```
model simple
uses "mmxad";

declarations
    id_win=1;id_textnumber=2;id_inputnumber=3;id_buttonok=4 !store ids for later
    N: real          !the number we are seeking
end-declarations

!user interface creation
XADcreatewindow      (id_win,50,50,300,200,"Need a number to continue")
    XADcreatetext    (id_win,id_textnumber,20,52,80,40,"Enter a number:")
    XADcreateinput   (id_win,id_inputnumber,100,50,100,20,"")
    XADcreatebutton  (id_win,id_buttonok,110,100,80,24,"OK")
```

First we create a window and assign the id `id_win` to it. Then we create three XAD objects: a descriptive text, an input field and a button, each with a different id. The ids should be longer rather than shorter and they should provide type information to make the model more readable and maintainable.

As an alternative to creating the window using several lines of Mosel code we may also create the window and associated objects using the IVE XAD Resource Editor (see the IVE help files for more information) and load them using the XAD procedure `XADloadresource`.

Next:

```
!event handler
procedure guievents(id:integer,event:integer)
    if      id=id_buttonok  and event=XAD_EVENT_PRESSED         then
        XADwindowclose(id_win)
    elif    id=id_win   and event=XAD_EVENT_WINDOW_CLOSED   then
        N:=real(XADinputgettext(id_inputnumber))
    elif    id=id_win   and event=XAD_EVENT_WINDOW_OPENED then
        XADsetfocus(id_inputnumber)
    end-if
end-procedure
```

The event handler is the core of an XAD program. All the interaction between the user and the GUI is reflected through the event handler. The event handler is a callback procedure that takes two arguments of type `integer`. XAD will call this procedure when an event occurs. The arguments are:

id:	`integer`	The id of the XAD object that generated the event
event:	`integer`	A number which denotes an event (*e.g.* `XAD_EVENT_PRESSED`)

Note that some events such as key presses and list selections carry *textual* information. If this information is needed, it can be retrieved in the event handler using the routine
`XADgeteventtext: string`

In our simple case, if the object `id_buttonok` is pressed then we close the window. When the window is closed, the text currently in the input object `id_inputnumber` is converted to a `real` number and assigned to `N`. Thirdly, for improved user interaction, we set the focus on the input object as soon as the window is opened.

Finally:

```
!set event handler
XADseteventcallback("guievents")

!open window - this is a "blocking" call
XADwindowopen(id_win)

!once window is closed execution continues here
writeln("N=",N)

end-model
```

The procedure which handles the events (the event handler) must be registered with XAD by calling `XADseteventcallback` with the event handler procedure *name* as the sole argument. You may, if you are using resource-loaded objects, create event callbacks by creating procedures named by the concatenation of the object name, window name and event type. For example the `Exit_Window_PRESSED` procedure as mentioned in section 11.1 is the event callback which will be called when the `Exit` button, resident in the window called `Window`, is pressed (receives the `PRESSED` event).

Once the event handler is in place we can open (show) the window. When closing the window (by pressing the *OK* button or clicking the close button or pressing *Esc*), Mosel will resume its execution from the statement immediately following `XADwindowopen`.

A few notes on using XAD with Mosel:

- XAD code may be placed anywhere in a Mosel model, however, the user is encouraged to separate the code implementing the visual functionality from other modeling/programming

statements, as needed.

- In order to run a model which uses *mmxad.dso*, the library must be present in the `dso` folder of the FICO™ Xpress installation (or pointed to be the `MOSEL_DOS` environment variable) and properly licensed through the license file. Xpress-IVE is *not* needed—XAD is an independent library which only requires the Mosel runtime library.

Chapter 3

Lifetime of XAD objects

An XAD object will exist until the model ends or until it is destroyed using `XADdestroy`. The object-specific routines (listed in the reference part of this document) can only operate on an object when the window containing the object is active (or, if the object is a window, only while the window is active). For this reason, the event `XAD_EVENT_WINDOW_OPENED` is of particular interest: it is the only chance to operate on the objects in a window before the window begins interacting with the user.

There are two distinct ways in which windows may behave:

1. When a window is *opened* (using `XADwindowopen`), the window takes control of the Mosel program. Until the window is *closed* (using `XADwindowclose`) the only way to execute code is from the event handler (directly, or by calling a subroutine). The statement following `XADwindowopen` will only be executed after the user has closed the window. This behavior can be used when user input is necessary before the program can continue or when the user interface is meant to control the Mosel program. This type of window is referred to as a *modal dialog*.

2. When a window is *shown* (using `XADwindowshow`), the window is displayed and Mosel continues immediately. A *shown* window should be used as an auxiliary window for monitoring program state, to display progress, etc. When the monitoring is complete (*e.g.* at the end of the Mosel run), the window may be *hidden*, using `XADwindowhide`. If the window is not hidden before the Mosel run ends, it will persist until the model is unloaded from memory (in Xpress-IVE this only happens when compiling or running a model). This type of window is referred to as a *modeless dialog*.

It is important to make a correct choice between the two behaviors. Experiment until the distinction is well understood, and keep in mind that for most purposes a window should be *opened* in order to allow it to take control of the program execution by generating events.

Chapter 4

Events

When the user operates on a window or object in a window, the action is reported through an event. The event handler callback procedure can be used to respond to such events. For example, when the user presses a button, the event handler callback procedures is called with two arguments: an integer representing the **id** of the button that triggered the event and another integer representing the event code `XAD_EVENT_PRESSED`. Or, when the user resizes a window by dragging its margins, the event is reported as a call to the event handler callback with the window id and `XAD_EVENT_WINDOW_RESIZED` as arguments.

Events specific to objects:

`XAD_EVENT_CHANGED`	Input, editor or scrollbar changed event.	p. 103
`XAD_EVENT_MENU`	Menu event.	p. 101
`XAD_EVENT_PRESSED`	Button pressed event.	p. 103
`XAD_EVENT_SELECTION`	Selection event.	p. 103
`XAD_EVENT_TIMER`	Timer event.	p. 101
`XAD_EVENT_WINDOW_CLOSED`	Window closed event.	p. 101
`XAD_EVENT_WINDOW_CLOSING`	Window closing event.	p. 102
`XAD_EVENT_WINDOW_HIDDEN`	Window hidden event.	p. 102
`XAD_EVENT_WINDOW_MOVED`	Window moved event.	p. 102
`XAD_EVENT_WINDOW_OPENED`	Window opened event.	p. 102
`XAD_EVENT_WINDOW_RESIZED`	Window resized event.	p. 102
`XAD_EVENT_WINDOW_SHOWN`	Window shown event.	p. 102

Generic events:

`XAD_EVENT_KEYDOWN`	Key pressed.	p. 136
`XAD_EVENT_KEYUP`	Key released.	p. 137
`XAD_EVENT_MOUSE_LEFTDBCLK`	Double click with left mouse button.	p. 137
`XAD_EVENT_MOUSE_LEFTDOWN`	Left mouse button pressed.	p. 137
`XAD_EVENT_MOUSE_LEFTUP`	Left mouse button released.	p. 137
`XAD_EVENT_MOUSE_MOVED`	Mouse moved.	p. 137

XAD_EVENT_MOUSE_RIGHTDBCLK	Double click with right mouse button.	p. 138
XAD_EVENT_MOUSE_RIGHTDOWN	Right mouse button pressed.	p. 138
XAD_EVENT_MOUSE_RIGHTUP	Right mouse button released.	p. 138

Chapter 5

Notes

When developing XAD applications in Xpress-IVE, do not use the *Stop* feature to end the program execution. This could interrupt the Mosel execution during a system call dealing with the user objects, leaving Mosel and IVE in a corrupt state. Always close all XAD windows instead of using *Stop*.

For more instances of XAD applications please refer to the set of examples that accompanies XAD.

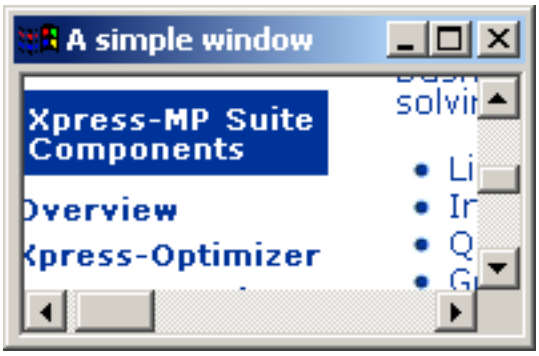
Chapter 6
XAD objects reference

XAD provides functionality to work with the following graphical objects:

- Browser (6.1)
- Button (6.2)
- Canvas (6.3)
- Check button (6.4)
- Drop list (6.5)
- Editor (6.6)
- Group (6.7)
- Input (6.8)
- List (6.9)
- Multilist (6.10)
- Progress bar (6.11)
- Radio button (6.12)
- Scroll bar (6.13)
- Tab (6.14)
- Text (6.15)
- Tree (6.16)
- Window (6.17)

6.1 Browser

An Internet Explorer-based web browser which can display any webpage.



Specific subroutines

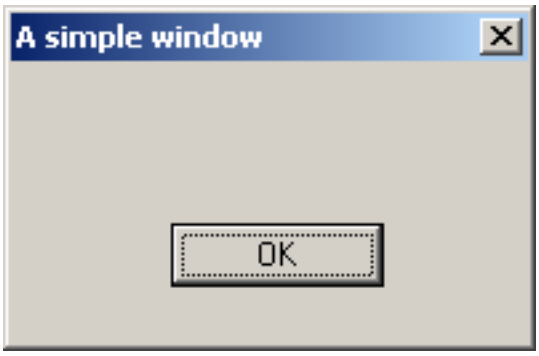
`XADcreatebrowser`, `XADbrowsergoto`

Specific events

None

6.2 Button

A regular push button.



Specific subroutines

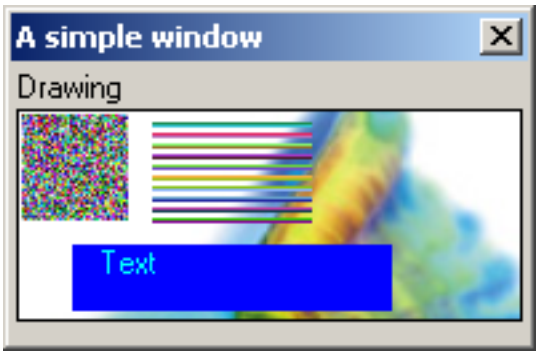
`XADcreatebutton`

Specific events

`XAD_EVENT_PRESSED`

6.3 Canvas

An intuitive surface for drawing anything using XAD.



Specific subroutines

`XADcreatecanvas`, `XADcanvasdrawarc`, `XADcanvasdrawbox`, `XADcanvasdrawchord`, `XADcanvasdrawellipse`, `XADcanvasdrawimage`, `XADcanvasdrawline`, `XADcanvasdrawpie`, `XADcanvasdrawpoint`, `XADcanvasdrawpolygon`, `XADcanvasdrawrectangle`, `XADcanvasdrawtext`, `XADcanvaserase`, `XADcanvasmap`, `XADcanvasrefresh`, `XADcanvassaveimage`, `XADcanvasunmap`

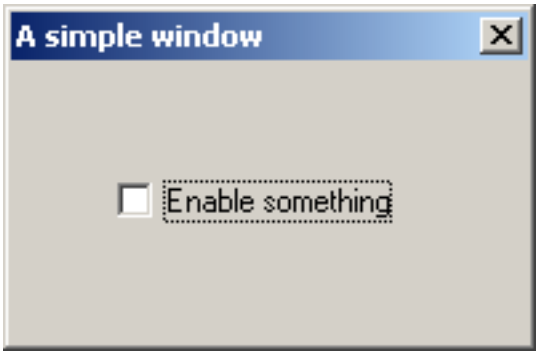
Specific events

None

Note Working with colors. XAD recognizes the following color constants: `XAD_BLACK`, `XAD_BLUE`, `XAD_CYAN`, `XAD_GREEN`, `XAD_MAGENTA`, `XAD_ORANGE`, `XAD_RED`, `XAD_WHITE`, `XAD_YELLOW` Use `XADcolor` to create any other color based on its red, green and blue components.

6.4 Check button

A button with two states: checked and unchecked. It is independent of any other objects.



Specific subroutines

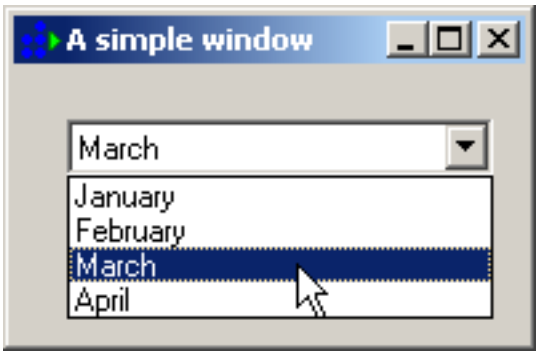
`XADcreatecheck`, `XADcheckgetstate`, `XADchecksetstate`

Specific events

`XAD_EVENT_PRESSED`

6.5 Drop list

A sortable list of strings, numbers, etc., which can be expanded or collapsed.



Specific subroutines

`XADcreatedroplist`, `XADdroplistadd`, `XADdroplistgetsel`, `XADdroplistselect`, `XADdroplistshow`

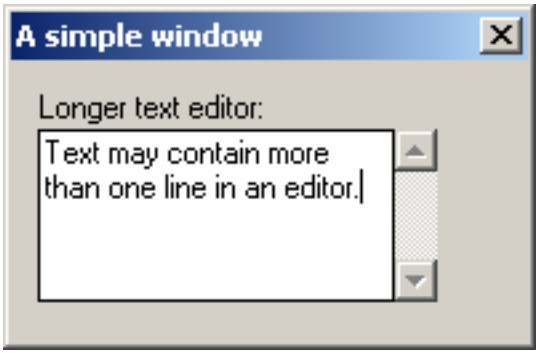
Specific events

`XAD_EVENT_SELECTION`

Note A list object generates an `XAD_EVENT_SELECTION` event when the user changes the selection. In order to find out which item was selected, call the function `XADgeteventtext`:string.

6.6 Editor

A field for editing multi-line text.



Specific subroutines

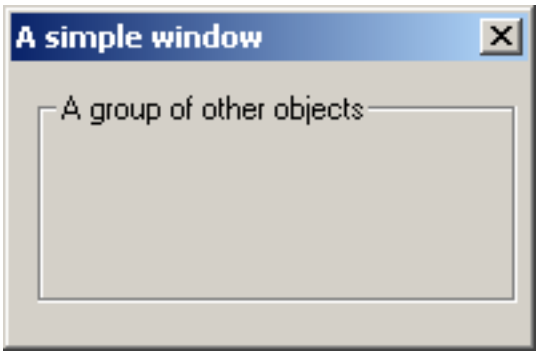
`XADcreateeditor`, `XADeditoraddtext`, `XADeditorgettext` `XADeditorload`, `XADeditorsave`, `XADeditorsettext`

Specific events

`XAD_EVENT_CHANGED`

6.7 Group

A thin frame surrounding a group of related objects.



Specific subroutines

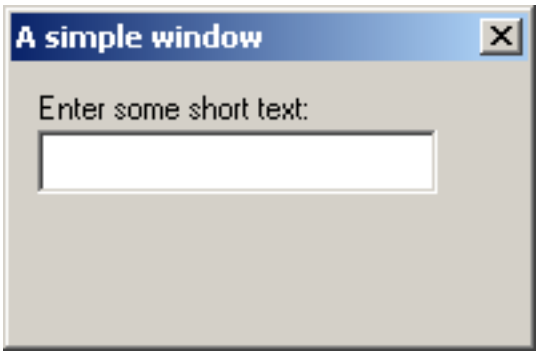
`XADcreategroup`

Specific events

None

6.8 Input

A single line input field.



Specific subroutines

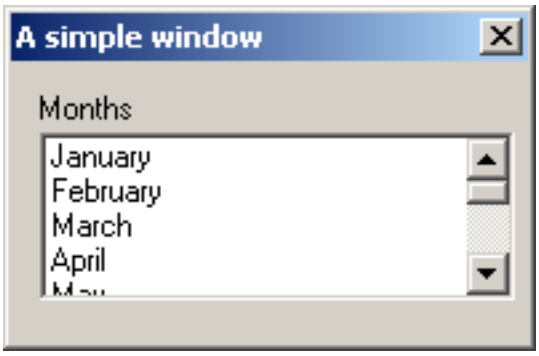
`XADcreateinput`, `XADinputgettext`, `XADinputsettext`

Specific events

`XAD_EVENT_CHANGED`

6.9 List

A sortable list of strings, numbers, etc.



Specific subroutines

`XADcreatelist`, `XADlistadd`, `XADlistgetsel`, `XADlistselect`, `XADlistshow`

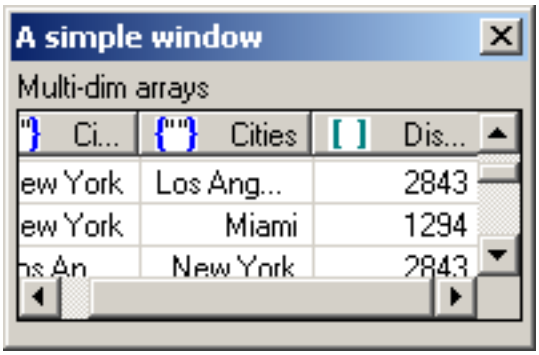
Specific events

`XAD_EVENT_SELECTION`

Note A list object generates an `XAD_EVENT_SELECTION` event when the user changes the selection. In order to find out which item was selected, call the function `XADgeteventtext`:string.

6.10 List with multiple columns

A sortable list of strings, numbers, etc. which can display multidimensional data.



Specific subroutines

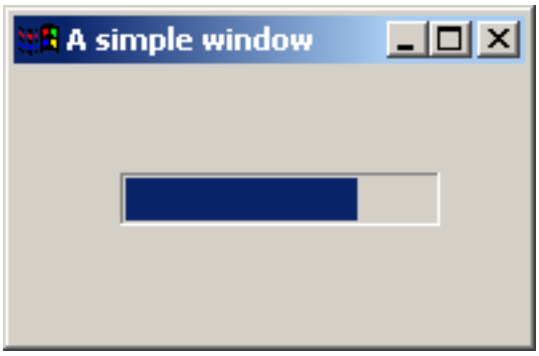
`XADcreatemultilist`, `XADmultilistrefresh`, `XADmultilistsetcolname`, `XADmultilistsetsize`, `XADmultilistsettext`, `XADmultilistshow`

Specific events

None

6.11 Progress bar

A visual progress indicator.



Specific subroutines

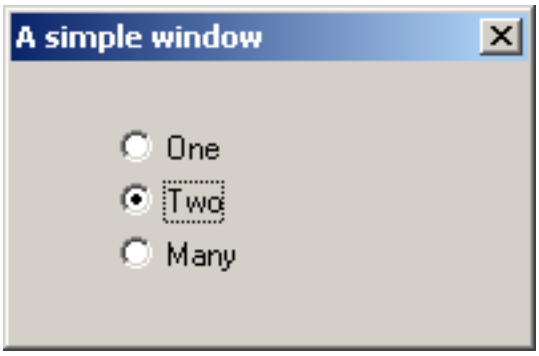
`XADcreateprogress`, `XADprogressset`

Specific events

None

6.12 Radio button

A button with two states: checked and unchecked. It can be used for mutually exclusive choices.



Specific subroutines

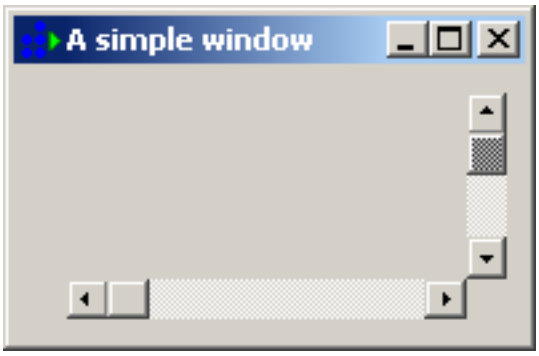
`XADcreateradio`, `XADradiogetstate`, `XADradiosetstate`

Specific events

`XAD_EVENT_PRESSED`

6.13 Scroll bars

A vertical or horizontal scrollbar for controlling position in a large document.



Specific subroutines

`XADcreatescrollbar`, `XADscrollbargetpos`, `XADscrollbarset`

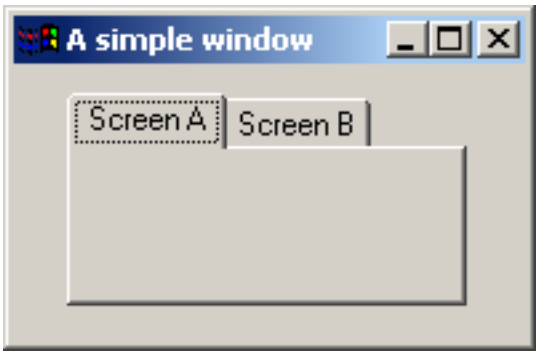
Specific events

`XAD_EVENT_CHANGED`

Note A scrollbar object generates an `XAD_EVENT_CHANGED` event when the user interacts with it. Use `XADscrollbargetpos` to obtain the new position.

6.14 Tab selectors

A notebook-style object for choosing among general categories (of other objects, usually).



Specific subroutines

`XADcreatetab`, `XADtabgettab`, `XADtabsettab`

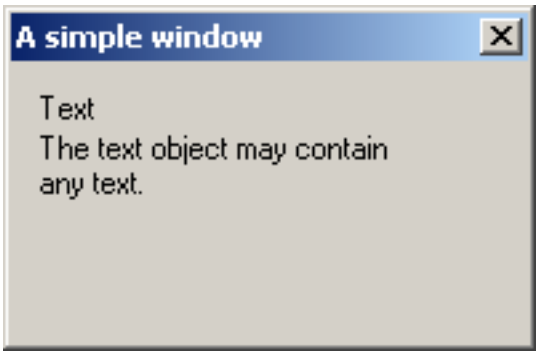
Specific events

`XAD_EVENT_SELECTION`

Note A tab object generates an `XAD_EVENT_SELECTION` event when the user changes the selection. In order to find out which tab was selected, call the function `XADgeteventtext`:string or use `XADtabgettab`.

6.15 Text

A multiline label for displaying text information.



Specific subroutines

`XADcreatetext`, `XADtextaddtext`, `XADtextgettext`, `XADtextsettext`

Specific events

None

6.16 Tree

A hierarchical tree display.

Specific subroutines

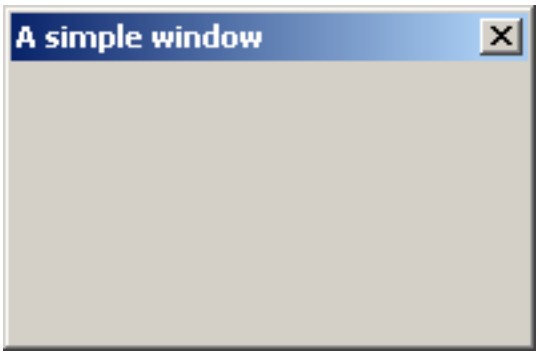
`XADcreatetree`, `XADtreeadd`, `XADtreereset`, `XADtreeexpand`

Specific events

XAD_EVENT_SELECTION

6.17 Window

The most important object; brings together all the other types of objects and allows interaction with the user.



Specific subroutines

XADcreatewindow, XADwindowaddmenu, XADwindowclose, XADwindowhide, XADwindowkeep, XADwindowopen, XADwindowsettimer, XADwindowshow

Specific events

XAD_EVENT_MENU, XAD_EVENT_TIMER, XAD_EVENT_WINDOW_CLOSED, XAD_EVENT_WINDOW_CLOSING, XAD_EVENT_WINDOW_HIDDEN, XAD_EVENT_WINDOW_MOVED, XAD_EVENT_WINDOW_OPENED, XAD_EVENT_WINDOW_RESIZED, XAD_EVENT_WINDOW_SHOWN

6.18 Subroutines specific to objects

XADcreatewindow

Purpose

Create a window

Synopsis

```
procedure XADcreatewindow(id:integer, x:integer, y:integer, w:integer,
      h:integer, name:string)
```

Arguments

`id`	Arbitrary unique id of the window
`x`	x coordinate (from left of screen) of the window in pixels
`y`	y coordinate (from top of screen) of the window in pixels
`w`	Width of the window in pixels
`h`	Height of the window in pixels
`name`	The title of the window

Related topics

`XADwindowopen`, `XADwindowclose`, `XADwindowshow`, `XADwindowhide`, `XADwindowkeep`, `XADwindowsettimer`

XADwindowopen

Purpose

Open a window.

Synopsis

```
procedure XADwindowopen(id:integer)
```

Argument

`id` Window identifier

Further information

Displays the window and surrenders execution control to its event handler callback.

Related topics

`XADwindowclose`

XADwindowclose

Purpose

Close a window.

Synopsis

```
procedure XADwindowclose(id:integer)
```

Argument

`id` Window identifier

Further information

Destroys an *opened* window.

Related topics

`XADwindowopen`

XADwindowshow

Purpose

Display a window.

Synopsis

```
procedure XADwindowshow(id:integer)
```

Argument

`id` Window identifier

Further information

Displays the window and execution resumes immediately.

Related topics

`XADwindowhide`

XADwindowhide

Purpose

Hide a window.

Synopsis

```
procedure XADwindowhide(id:integer)
```

Argument

`id` Window identifier

Further information

Destroys a *shown* window.

Related topics

`XADwindowshow`

XADwindowkeep

Purpose

Keep a window.

Synopsis

```
procedure XADwindowkeep
```

Further information

Should be used only when dealing with the `XAD_EVENT_WINDOW_CLOSING` event, if the intent is to cancel the closing of the window. For example, if an exit confirmation dialog is shown and the user changes his mind, call this procedure to prevent the window from closing.

Related topics

`XADwindowhide`

XADwindowsettimer

Purpose

(Re)set a timer.

Synopsis

```
procedure XADwindowsettimer(id:integer, msec:integer)
```

Arguments

`id`	Window identifier
`msec`	Timer interval in milliseconds

Further information

Sets or resets a timer that generates `XAD_EVENT_TIMER` events. If ***msec*** is a positive integer, the event will be generated every ***msec*** milliseconds. If ***msec*** is 0 or negative, the timer is reset (timer events are no longer generated)

XADwindowaddmenu

Purpose

Add a dropdown menu to the window.

Synopsis

```
procedure XADwindowaddmenu(id:integer, title:string, items:string)
procedure XADwindowaddmenu(id:integer, title:string, itemset:  set of
      string)
```

Arguments

`id` Window identifier

`title` Menu title (e.g. File, Edit)

`items` Comma-separated list of strings representing the menu items

`itemset` Set of strings representing the menu items

Example

```
XADwindowaddmenu(id_win, "", "")  !pass empty string to clear existing menu, i
XADwindowaddmenu(id_win, "&File", "&New,&Open,&Save,XADseparator,E&xit")
XADwindowaddmenu(id_win, "&Help", "&Help on MyApp,&About...")
```

Further information

Respond to menu selections by checking the `XAD_EVENT_MENU` event.

Use the ampersand character & to create a Alt+letter hotkey for each menu item.

Use the string "***XADseparator***" to insert a line separator in the menu.

Pass an empty ***title*** to reset and destroy the menu. Then rebuild the entire menu.

XADcreatetext

Purpose

Create a text object.

Synopsis

```
procedure XADcreatetext(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
```

Arguments

`wid`	id of the window containing the text
`id`	Text identifier
`x`	x coordinate (from left of window) of the text in pixels
`y`	y coordinate (from top of window) of the text in pixels
`w`	Width of the text in pixels
`h`	Height of the text in pixels
`text`	Contents of the text object (use `\r\n` to break lines)

Related topics

`XADtextaddtext`, `XADtextgettext`, `XADtextsettext`

XADtextaddtext

Purpose

Add text to a text object.

Synopsis

```
procedure XADtextaddtext(id:integer, text:string)
```

Arguments

`id`	Text identifier
`text`	Text to be added

Further information

Appends the given ***text*** to the object.

Related topics

`XADtextgettext`, `XADtextsettext`

XADtextsettext

Purpose

Set text of a text object.

Synopsis

```
procedure XADtextsettext(id:integer, text:string)
```

Arguments

`id`	Text identifier
`text`	New text

Further information

Replaces the text of the object with the new ***text***.

Related topics

`XADtextaddtext`, `XADtextgettext`

XADtextgettext

Purpose

Get the text of a text object.

Synopsis

```
function XADtextgettext(id:integer):string
```

Argument

`id` Text identifier

Return value

Text of the object.

Further information

Obtains the text currently shown by the object.

Related topics

`XADtextaddtext`, `XADtextsettext`

XADcreatebutton

Purpose

Create a button object.

Synopsis

```
procedure XADcreatebutton(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
```

Arguments

`wid`	id of the window containing the button
`id`	Button identifier
`x`	x coordinate (from left of window) of the button in pixels
`y`	y coordinate (from top of window) of the button in pixels
`w`	Width of the button in pixels
`h`	Height of the button in pixels
`name`	The name of the button

XADcreateinput

Purpose

Create an input object.

Synopsis

```
procedure XADcreateinput(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, text:string)
```

Arguments

`wid`	id of the window containing the input
`id`	Input identifier
`x`	x coordinate (from left of window) of the input in pixels
`y`	y coordinate (from top of window) of the input in pixels
`w`	Width of the input in pixels
`h`	Height of the input in pixels
`text`	Initial content of the input object

Related topics

`XADinputgettext`, `XADinputsettext`

XADinputsettext

Purpose

Set the text of an input object.

Synopsis

```
procedure XADinputsettext(id:integer, text:string)
```

Arguments

`id`	Input identifier
`text`	New text

Further information

Replaces the ***text*** of the object with the new text.

Related topics

`XADinputgettext`

XADinputgettext

Purpose

Get the text of an input object.

Synopsis

```
function XADinputgettext(id:integer):string
```

Argument

`id` Input identifier

Return value

Text of the object.

Further information

Obtains the text currently shown by the object.

Related topics

`XADinputsettext`

XADcreateeditor

Purpose

Create an editor object.

Synopsis

```
procedure XADcreateeditor(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, text:string)
```

Arguments

`wid`	id of the window containing the editor
`id`	Editor identifier
`x`	x coordinate (from left of window) of the editor in pixels
`y`	y coordinate (from top of window) of the editor in pixels
`w`	Width of the editor in pixels
`h`	Height of the editor in pixels
`text`	Contents of the editor object (use `\r\n` to break lines)

Related topics

`XADeditoraddtext`, `XADeditorload`, `XADeditorsave`, `XADeditorsettext`, `XADeditorgettext`

XADeditoraddtext

Purpose

Add text to an editor.

Synopsis

```
procedure XADeditoraddtext(id:integer, text:string)
```

Arguments

`id`	Editor identifier
`text`	Text to be added

Further information

Appends the given ***text*** to the object.

Related topics

`XADeditorsettext`, `XADeditorgettext`

XADeditorload

Purpose

Load a file into an editor.

Synopsis

```
procedure XADeditorload(id:integer, file:string)
```

Arguments

`id`	Editor identifier
`file`	File name

Further information

Load the specified ***file*** into the editor, replacing its previous content.

Related topics

`XADeditorsave`

XADeditorsave

Purpose

Save editor contents into a file.

Synopsis

```
procedure XADeditorsave(id:integer, file:string)
```

Arguments

`id`	Editor identifier
`file`	File name

Further information

Saves the contents of the editor to the specified ***file***.

Related topics

`XADeditorload`

XADeditorsettext

Purpose

Set the text of an editor.

Synopsis

```
procedure XADeditorsettext(id:integer, text:string)
```

Arguments

`id`	Editor identifier
`text`	New text

Further information

Replaces the text of the object with the new ***text***.

Related topics

`XADeditoraddtext`, `XADeditorgettext`

XADeditorgettext

Purpose

Get the text from an editor.

Synopsis

```
function XADeditorgettext(id:integer):string
```

Argument

`id` Editor identifier

Return value

Text of the object.

Further information

Obtains the text currently shown by the object.

Related topics

`XADeditoraddtext`, `XADeditorsettext`,

XADcreatecheck

Purpose

Create a check button.

Synopsis

```
procedure XADcreatecheck(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
```

Arguments

`wid`	id of the window containing the check
`id`	Check identifier
`x`	x coordinate (from left of window) of the check in pixels
`y`	y coordinate (from top of window) of the check in pixels
`w`	Width of the check in pixels
`h`	Height of the check in pixels
`name`	Description of the check object

Related topics

`XADchecksetstate`, `XADcheckgetstate`

XADchecksetstate

Purpose

Set the state of a check button.

Synopsis

```
procedure XADchecksetstate(id:integer, state:boolean)
```

Arguments

`id` Check identifier

`state` Button state.

- `true` checked
- `false` unchecked

Further information

Sets the ***state*** of the check to checked or unchecked.

Related topics

`XADcheckgetstate`

XADcheckgetstate

Purpose

Get the state of a check button.

Synopsis

```
function XADcheckgetstate(id:integer):boolean
```

Argument

`id` Check identifier

Return value

`true` Button is checked

`false` Button is unchecked

Further information

Obtains the state of the check (`true` for checked, `false` for unchecked).

Related topics

`XADchecksetstate`

XADcreateradio

Purpose

Create a radio button.

Synopsis

```
procedure XADcreateradio(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
```

Arguments

`wid`	id of the window containing the radio
`id`	Radio identifier
`x`	x coordinate (from left of window) of the radio in pixels
`y`	y coordinate (from top of window) of the radio in pixels
`w`	Width of the radio in pixels
`h`	Height of the radio in pixels
`name`	Name of the radio button object

Related topics

`XADradiosetstate`, `XADradiogetstate`

XADradiosetstate

Purpose

Get the state of a radio button.

Synopsis

```
procedure XADradiosetstate(id:integer, state:boolean)
```

Arguments

`id` Radio identifier

`state` Button state.

- `true` selected
- `false` unselected

Further information

Sets the ***state*** of the radio to selected (true) or unselected (false).

Related topics

`XADradiogetstate`

XADradiogetstate

Purpose

Get the state of a radio button.

Synopsis

```
function XADradiogetstate(id:integer):boolean
```

Arguments

`id` Radio identifier

Return value

`true` Radio button is selected

`false` Radio button is unselected

Further information

Obtains the state of the radio (true for selected, false for unselected).

Related topics

`XADradiosetstate`

XADcreategroup

Purpose

Create a group object.

Synopsis

```
procedure XADcreategroup(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
```

Arguments

`wid`	id of the window containing the group
`id`	Group identifier
`x`	x coordinate (from left of window) of the group in pixels
`y`	y coordinate (from top of window) of the group in pixels
`w`	Width of the group in pixels
`h`	Height of the group in pixels
`name`	Description of the group

XADcreatelist

Purpose

Create a list object.

Synopsis

```
procedure XADcreatelist(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
procedure XADcreatelist(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string, sorted:boolean)
```

Arguments

`wid`	id of the window containing the list
`id`	List identifier
`x`	x coordinate (from left of window) of the list in pixels
`y`	y coordinate (from top of window) of the list in pixels
`w`	Width of the list in pixels
`h`	Height of the list in pixels
`name`	String containing comma-separated list items
`sorted`	If **true**, the list will be sorted and remain sorted after adding elements

Related topics

`XADlistadd`, `XADlistgetsel`, `XADlistshow`

XADlistadd

Purpose

Add an item to a list

Synopsis

```
procedure XADlistadd(id:integer, item:string)
```

Arguments

`id`	List identifier
`item`	Item to add

Further information

Appends the ***item*** to the list.

Related topics

`XADlistgetsel`, `XADlistshow`

XADlistgetsel

Purpose

Get the selected item from a list.

Synopsis

```
function XADlistgetsel(id:integer):string
```

Arguments

`id` List identifier

Return value

Selected item.

Further information

Obtains the item currently selected.

Related topics

`XADlistadd`, `XADlistshow`, `XADlistselect`

XADlistselect

Purpose

Select a list item

Synopsis

```
procedure XADlistselect(id:integer, item:string)
```

Arguments

`id`	List identifier
`item`	Item to select

Further information

If found, selects the ***item***.

Related topics

`XADlistgetsel`, `XADlistshow`

XADlistshow

Purpose

Show a list.

Synopsis

```
procedure XADlistshow(id:integer, items:set)
```

Arguments

`id` List identifier

`items` Items to display

Further information

Fills the list with the contents of the given set of ***items***. Mosel will try to infer the type of the set and display accordingly. For example showing a *set of mpvar* will display the solution values and reduced costs for each item.

Related topics

`XADlistadd`, `XADlistgetsel`

XADcreatedroplist

Purpose

Create a droplist object.

Synopsis

```
procedure XADcreatedroplist(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string)
procedure XADcreatedroplist(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, name:string, sorted:boolean)
```

Arguments

`wid`	id of the window containing the droplist
`id`	Droplist identifier
`x`	x coordinate (from left of window) of the droplist in pixels
`y`	y coordinate (from top of window) of the droplist in pixels
`w`	Width of the droplist in pixels
`h`	Height of the droplist in pixels
`name`	String containing comma-separated droplist items
`sorted`	If **true**, the droplist will be sorted and remain sorted after adding elements

Related topics

`XADdroplistadd`, `XADdroplistgetsel`, `XADdroplistshow`

XADdroplistadd

Purpose

Add an item to a droplist

Synopsis

```
procedure XADdroplistadd(id:integer, item:string)
```

Arguments

`id`	Droplist identifier
`item`	Item to add

Further information

Appends the ***item*** to the droplist.

Related topics

`XADdroplistgetsel`, `XADdroplistshow`

XADdroplistgetsel

Purpose

Get the selected item from a droplist.

Synopsis

```
function XADdroplistgetsel(id:integer):string
```

Arguments

`id` Droplist identifier

Return value

Selected item.

Further information

Obtains the item currently selected.

Related topics

`XADdroplistadd`, `XADdroplistshow`, `XADdroplistselect`

XADdroplistselect

Purpose

Select a droplist item

Synopsis

```
procedure XADdroplistselect(id:integer, item:string)
```

Arguments

`id`	Droplist identifier
`item`	Item to select

Further information

If found, selects the ***item***.

Related topics

`XADdroplistgetsel`, `XADdroplistshow`

XADdroplistshow

Purpose

Show a droplist.

Synopsis

```
procedure XADdroplistshow(id:integer, items:set)
```

Arguments

`id` Droplist identifier

`items` Items to display

Further information

Fills the droplist with the contents of the given set of ***items***. Mosel will try to infer the type of the set and display accordingly. For example showing a *set of mpvar* will display the solution values and reduced costs for each item.

Related topics

`XADdroplistadd`, `XADdroplistgetsel`

XADcreateprogress

Purpose

Create a progress bar.

Synopsis

```
procedure XADcreateprogress(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer)
```

Arguments

`wid`	id of the window containing the progress
`id`	Progress identifier
`x`	x coordinate (from left of window) of the progress in pixels
`y`	y coordinate (from top of window) of the progress in pixels
`w`	Width of the progress in pixels
`h`	Height of the progress in pixels

Related topics

`XADprogressset`

XADprogressset

Purpose

Set the progress state.

Synopsis

```
procedure XADprogressset(id:integer, minval:real, maxval:real,
       current:real)
```

Arguments

`id` Progress identifier

`minval` Lower bound

`maxval` Upper bound

`current` Current value

Further information

The progress will show how far ***current*** is between ***minval*** and ***maxval***.

XADcreatetab

Purpose

Create a tab selector object.

Synopsis

```
procedure XADcreatetab(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, tabset:set of string)
procedure XADcreatetab(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, tabs:string)
```

Arguments

`wid`	id of the window containing the tab
`id`	Tab identifier
`x`	x coordinate (from left of window) of the tab in pixels
`y`	y coordinate (from top of window) of the tab in pixels
`w`	Width of the tab in pixels
`h`	Height of the tab in pixels
`tabset`	Set of strings containing the tab names
`tabs`	String containing an ordered, comma-separated list of tab names

XADtabgettab

Purpose

Get the current tab selection.

Synopsis

```
function XADtabgettab(id:integer):string
```

Argument

`id` Tab identifier

Return value

Selected item.

XADtabsettab

Purpose

Select a given tab.

Synopsis

```
procedure XADtabsettab(id:integer, tab:string)
```

Arguments

`id`	Tab identifier
`tab`	tab to be selected

XADcreatemultilist

Purpose

Create a multilist object.

Synopsis

```
procedure XADcreatemultilist(wid:integer, id:integer, name:string,
      x:integer, y:integer, w:integer, h:integer)
```

Arguments

`wid`	id of the window containing the multilist
`id`	Multilist identifier
`x`	x coordinate (from left of window) of the multilist in pixels
`y`	y coordinate (from top of window) of the multilist in pixels
`w`	Width of the multilist in pixels
`h`	Height of the multilist in pixels

Related topics

`XADmultilistshow`, `XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistsettext`

XADmultilistshow

Purpose

Load and display multi-dimensional arrays or a set in a multilist.

Synopsis

```
procedure XADmultilistshow(id:integer, items:  array)
```

Arguments

`id` Multilist identifier

`items` Comma-separated list of array names to display in the multilist, or just one array name.

Example

```
XADmultilistshow(id_multi, "Units, Salesforce, NetProfit")
```

Further information

Clears the content of the multilist and fills it with the given array of ***items*** (integers, reals, strings, booleans, mpvars or linctrs). If more than one array is given, they must have the exact same index sets and shape.

Related topics

`XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistsettext`

XADmultilistsetsize

Purpose

(Re)set the size of a multilist.

Synopsis

```
procedure XADmultilistsetsize(id:integer, rows:integer, columns:integer)
procedure XADmultilistsetsize(id:integer, rows:integer, columns:integer,
      callback:string)
```

Arguments

`id` Multilist identifier

`rows` Number of rows

`columns` Number of columns

`callback` Name of callback function for requesting element values

Example 1

A static multilist

```
procedure SetUpStaticList
  XADmultilistsetsize(id_multidynamic, 10000, 8)
  forall(i in 1..8) XADmultilistsetcolname(id_multidynamic, i, "Col "+i)
  forall(i in 1..10000, j in 1..8) XADmultilistsettext(id_multidynamic, i, j,
  XADmultilistrefresh(id_multidynamic)
end-procedure
```

Example 2

A dynamic multilist

```
function ElementGenerator(id:integer, row:integer, col:integer):string
  returned:=""+row+","+col    !Generate element based on row, col
end-function

procedure SetUpDynamicList
  !Set up the multilist to ask for its items only when they are needed
  !a multilist with 10,000 rows and 8 columns which uses a callback
  XADmultilistsetsize(id_multidynamic, 10000, 8, "ElementGenerator")

  !Column names
  forall(i in 1..8) XADmultilistsetcolname(id_multidynamic, i, "Col "+i)
end-procedure
```

Further information

Clears the content of the multilist and prepares to hold the given number of ***rows*** and ***columns***. Note that row and column indices start with 1.

Related topics

`XADmultilistsetcolname`, `XADmultilistsettext`

XADmultilistsetcolname

Purpose

Set multilist column names.

Synopsis

```
procedure XADmultilistsetcolname(id:integer, column:integer, name:string)
```

Arguments

`id` Multilist identifier

`column` Column index

`name` Column name

Further information

The header of the ***column*** will display the new ***name***.

Related topics

`XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistrefresh`

XADmultilistsettext

Purpose

Set a multilist item.

Synopsis

```
procedure XADmultilistsettext(id:integer, row:integer, column:integer,
      text:string)
```

Arguments

`id`	Multilist identifier
`column`	Column index
`name`	Column name
`text`	New text of the item

Further information

The item at the given ***row*** and ***column*** will hold the new ***text***.

Related topics

`XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistrefresh`

XADmultilistgetsel

Purpose

Retrieve the selected multilist item.

Synopsis

```
procedure XADmultilistgetsel(id:integer, list:set of integer)
```

Arguments

`id`	The id of the multilist object
`list`	The set of rows selected in the multilist

Related topics

`XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistrefresh`

XADmultilistsetcolors

Purpose

Set the multilist item colour for a particular cell.

Synopsis

```
procedure XADmultilistsetcolors(id:integer, row:integer, column:integer,
      bgcolour:integer, fgcolour:integer)
```

Arguments

`id` The id of the multilist object

`row` The row of the cell to colour

`column` The column of the cell to colour

`bgcolour` The background colour as a 24bit colour value

`fgcolour` The foreground colour as a 24bit colour value

Related topics

`XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistrefresh`

XADmultilistrefresh

Purpose

Update the visual display (usually after many `XADmultilistsettext` operations).

Synopsis

```
procedure XADmultilistrefresh(id:integer)
```

Argument

`id` Multilist identifier

Related topics

`XADmultilistsetsize`, `XADmultilistsetcolname`, `XADmultilistsettext`

XADcreatecanvas

Purpose

Create a canvas object.

Synopsis

```
procedure XADcreatecanvas(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer)
```

Arguments

`wid`	id of the window containing the canvas
`id`	Canvas identifier
`x`	x coordinate (from left of window) of the object in pixels
`y`	y coordinate (from top of window) of the object in pixels
`w`	Width of the object in pixels
`h`	Height of the object in pixels

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawellipse`, `XADcanvaserase`, `XADcanvasrefresh`, `XADcanvasdrawimage`, `XADcanvassaveimage`, `XADcanvasdrawline`, `XADcanvasdrawpoint`, `XADcanvasdrawrectangle`, `XADcanvasdrawtext`, `XADcanvasmap`, `XADcanvasunmap`, `XADcolor`

XADcanvasdrawbox

Purpose

Draw a box on a canvas.

Synopsis

```
procedure XADcanvasdrawbox(id:integer, x:real, y:real, w:real, h:real,
      color1:integer, color2:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate (from left of canvas)
`y`	y coordinate (from top of canvas)
`w`	width of box
`h`	height of box
`color1`	Border color
`color2`	Fill color

Further information

Draws a solid rectangle of color ***color2*** with a one-pixel margin of color ***color1*** of width ***w*** and height ***h*** at coordinates (***x***,***y***).

Related topics

`XADcanvasdrawline`, `XADcanvasdrawpoint`, `XADcanvasdrawrectangle`, `XADcanvasdrawtext`, `XADcolor`

XADcanvasdrawellipse

Purpose

Draw an ellipse on a canvas.

Synopsis

```
procedure XADcanvasdrawellipse(id:integer, x:real, y:real, w:real, h:real,
      color1:integer, color2:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate (from left of canvas)
`y`	y coordinate (from top of canvas)
`w`	width of ellipse
`h`	height of ellipse
`color1`	Border color
`color2`	Fill color

Further information

Draws a solid ellipse of color ***color2*** with a one-pixel margin of color ***color1*** of width ***w*** and height ***h*** at coordinates (***x***,***y***).

Related topics

`XADcanvasdrawline`, `XADcanvasdrawpoint`, `XADcanvasdrawrectangle`, `XADcanvasdrawtext`, `XADcolor`

XADcanvaserase

Purpose

Erase a canvas.

Synopsis

```
procedure XADcanvaserase(id:integer, color:integer)
```

Arguments

`id` Canvas identifier

`color` Color selection

Further information

Clears the contents of the canvas using the specified ***color***

Related topics

`XADcanvasrefresh`

XADcanvasrefresh

Purpose

Redraw a canvas.

Synopsis

```
procedure XADcanvasrefresh(id:integer)
```

Argument

`id` Canvas identifier

Further information

Updates the visual content of the canvas. Simply drawing on a canvas does not update its appearance (to save time). Only call `XADcanvasrefresh` when necessary.

Related topics

`XADcanvaserase`

XADcanvasdrawimage

Purpose

Draw an image from file.

Synopsis

```
procedure XADcanvasdrawimage(id:integer, x:real, y:real, file:  string)
procedure XADcanvasdrawimage(id:integer, x:real, y:real, w:integer,
      h:integer, file:  string)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`w`	Width
`h`	Height
`file`	File name

Further information

Draws the image from the given ***file*** at coordinates (***x***,***y***). The file can be of type `.bmp`, `.jpg`, `.gif`, or `.png`. If height and width are given, then the image is rescaled to fit in a rectangle of width ***w*** and height ***h***. If the string is `"xadimg:imgname"`, a previously saved ***imgname*** (see above) will be drawn.

Related topics

`XADcanvassaveimage`

XADcanvassaveimage

Purpose

Save an image to memory (to avoid loading a file repeatedly).

Synopsis

```
procedure XADcanvassaveimage(id:integer, x:real, y:real, w:integer,
      h:integer, imgname:  string)
procedure XADcanvassaveimage(id:integer, filename:string, imgname:  string)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`w`	Width
`h`	Height
`filename`	File containing an image (`.bmp`, `.jpg`, `.gif`, or `.png`)
`imgname`	Image name

Further information

Saves what is currently drawn at ***x***,***y***,***w***,***h*** or the image in ***filename*** under the identifier ***imgname***. ***imgname*** can then be drawn anywhere else using `XADcanvasdrawimage`.

Related topics

`XADcanvasdrawimage`

XADcanvasdrawline

Purpose

Draw a line on a canvas.

Synopsis

```
procedure XADcanvasdrawline(id:integer, x1:real, y1:real, x2:real, y2:real,
      color:integer)
procedure XADcanvasdrawline(id:integer, x1:real, y1:real, x2:real, y2:real,
      color:integer, width:  integer)
```

Arguments

`id`	Canvas identifier
`x1`	x start coordinate
`y1`	y start coordinate
`x2`	x end coordinate
`y2`	y end coordinate
`color`	Color
`width`	Width

Further information

Draws a line ***width***-pixels wide from (***x1***,***y1***) to (***x2***,***y2***) with the given ***color***. The default width is one pixel.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawpoint`, `XADcanvasdrawrectangle`, `XADcanvasdrawtext`, `XADcolor`

XADcanvasdrawpoint

Purpose

Draw a point on a canvas.

Synopsis

```
procedure XADcanvasdrawpoint>(id:integer, x:real, y:real, color:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`color`	Color

Further information

Draws a pixel at (***x***,***y***) with the given ***color***.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawline`, `XADcanvasdrawrectangle`, `XADcanvasdrawtext`, `XADcolor`

XADcanvasdrawrectangle

Purpose

Draw a rectangle on a canvas.

Synopsis

```
procedure XADcanvasdrawrectangle>(id:integer, x:real, y:real, w:real,
      h:real, color:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`w`	Width
`h`	Height
`color`	Color

Further information

Draws a solid rectangle of width ***w*** and height ***h*** at coordinates (***x***,***y***) with the given ***color***.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawline`, `XADcanvasdrawpoint`, `XADcanvasdrawtext`, `XADcolor`

XADcanvasdrawpolygon

Purpose

Draw a polygon on a canvas.

Synopsis

```
procedure XADcanvasdrawpolygon(id:integer, xs:array of real, ys:array of
      real, color1:integer, color2:integer)
```

Arguments

`id` Canvas identifier

`xs` Array of x coordinates

`ys` Array of y coordinates

`color1` Border color

`color2` Fill color

Example

```
declarations
  POINTS=1..3
  xs,ys:array(POINTS) of real
end-declarations

xs::[50, 100, 30]
ys::[50, 200, 80]

...
XADcanvasdrawpolygon(id_canvas,xs,ys,XAD_RED,XAD_BLACK)
```

Further information

Draws a using x and y coordinates taken from the ***xs*** and ***ys*** arrays. ***xs*** and ***ys*** must be indexed by the same index set. See example.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawline`, `XADcanvasdrawpoint`, `XADcanvasdrawtext`, `XADcolor`

XADcanvasdrawpie

Purpose

Draw an elliptical pie ***slice*** on a canvas.

Synopsis

```
procedure XADcanvasdrawpie(id:integer, x:real, y:real, w:real, h:real,
      start:real, end:real, color1:integer, color2:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`w`	Width of bounding rectangle
`h`	Height of bounding rectangle
`start`	Clockwise start percentage [0-100]
`end`	Clockwise end percentage [0-100]
`color1`	Border color
`color2`	Fill color

Example

```
declarations
  sizes: array(1..4) of real
  colors: array(1..4) of integer
  base: real
end-declarations
colors::[XAD_RED,XAD_GREEN,XAD_MAGENTA,XAD_BLACK]
forall(o in 1..4) sizes(o):=random*35; ! between 0-35%

...

base:=0
forall(o in 1..4) do
  !circular pie
  XADcanvasdrawpie(id_canvas,100,150,100,100,base,sizes(o),colors(o),colors(o)
  base+=sizes(o)
end-do
```

Further information

Draws a pie slice bounded by the given rectangle. The "angle" of the slice must be between [0-100] (percentage) and is measured clockwise beginning at 12 o'clock. See example.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawarc`, `XADcanvasdrawchord`

XADcanvasdrawarc

Purpose

Draw an elliptical arc on a canvas.

Synopsis

```
procedure XADcanvasdrawarc(id:integer, x:real, y:real, w:real, h:real,
      start:real, end:real, color:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`w`	Width of bounding rectangle
`h`	Height of bounding rectangle
`start`	Clockwise start percentage [0-100]
`end`	Clockwise end percentage [0-100]
`color1`	Color

Further information

Draws an elliptical arc bounded by the given rectangle. The "angle" of the arc must be between [0-100] (percentage) and is measured clockwise beginning at 12 o'clock.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawpie`, `XADcanvasdrawchord`

XADcanvasdrawchord

Purpose

Draw an elliptical chord on a canvas.

Synopsis

```
procedure XADcanvasdrawchord(id:integer, x:real, y:real, w:real, h:real,
      start:real, end:real, color1:integer, color2:integer)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`w`	Width of bounding rectangle
`h`	Height of bounding rectangle
`start`	Clockwise start percentage [0-100]
`end`	Clockwise end percentage [0-100]
`color1`	Border color
`color2`	Fill color

Example

```
declarations
  sizes: array(1..4) of real
  colors: array(1..4) of integer
  base: real
end-declarations
colors::[XAD_RED,XAD_GREEN,XAD_MAGENTA,XAD_BLACK]
forall(o in 1..4) sizes(o):=random*35; ! between 0-35%

...

base:=0
forall(o in 1..4) do
  !circular pie
  XADcanvasdrawchord(id_canvas,100,150,100,100,base,sizes(o),colors(o),colors(
  base+=sizes(o)
end-do
```

Further information

Draws a elliptical chord bounded by the given rectangle. The "angle" of the chord must be between [0-100] (percentage) and is measured clockwise beginning at 12 o'clock. See example.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawarc`, `XADcanvasdrawpie`

XADcanvasdrawtext

Purpose

Draw text on a canvas.

Synopsis

```
procedure XADcanvasdrawtext(id:integer, x:real, y:real, text:string,
      color:integer)
procedure XADcanvasdrawtext(id:integer, x:real, y:real, text:string,
      color:integer, fontsize:integer, alignment:integer, fontname:string)
```

Arguments

`id`	Canvas identifier
`x`	x coordinate
`y`	y coordinate
`text`	Text
`color`	Color
`fontsize`	Font size
`alignment`	Vertical and horizontal alignment
`fontname`	Font name

Further information

Draws ***text*** with font ***fontname*** of size ***fontsize*** at coordinates (***x***,***y***) with the given ***color*** and ***alignment***. If font name and size are not given then the output uses Arial 10pt. The alignment is the sum of one of `XAD_CENTERH`, `XAD_LEFT`, and `XAD_RIGHT` (horizontal alignment) plus one of `XAD_CENTERV`, `XAD_TOP`, and `XAD_BOTTOM` (vertical alignment). `XAD_DEFAULT` can be used instead to specify upper left alignment.

Related topics

`XADcanvasdrawbox`, `XADcanvasdrawline`, `XADcanvasdrawpoint`, `XADcanvasdrawrectangle`, `XADcolor`

XADcanvasmap

Purpose

Map the coordinate space of a canvas.

Synopsis

```
procedure XADcanvasmap(id:integer, x1:real, y1:real, x2:real, y2:real,
      x1new:real, y1new:real, x2new:real, y2new:real)
```

Arguments

`id`	Canvas identifier
`x1`	x start coordinate
`y1`	y start coordinate
`x2`	x end coordinate
`y2`	y end coordinate
`x1new`	New x start coordinate
`y1new`	New y start coordinate
`x2new`	New x end coordinate
`y2new`	New y end coordinate

Further information

Transforms the coordinate space of the canvas. Normally, (0,0) represents the upper left corner of the canvas and widths and heights are measured in pixels. By remapping the coordinate space, transformations no longer need to be applied to coordinates when drawing. For example, if the canvas has a width of 300 and a height of 200 and if we want to draw a graph from -1 to +1 on the x axis and -2 to +2 on the y axis in a region of the original canvas, we could write, say:

```
XADcanvasmap(id_canvas, 50, 150, 250, 50, -1, -2, 1, 2)
```

before plotting the points on the graph. Note that this transformation reverts the direction of the y axis and plots everything between (-1,1) and (-2,2) in the portion of the canvas between pixel coordinates (50,50) and (250,150). The following figure clarifies the effect of the `XADcanvasmap` call:

After calling this procedure, coordinates should be given in the *red* intervals. XAD will map them correctly onto the canvas, based on the *black* intervals.

Related topics

`XADcanvasunmap`

XADcanvasunmap

Purpose

Revert to default mapping for a canvas.

Synopsis

```
procedure XADcanvasunmap(id:integer)
```

Argument

`id` Canvas identifier

Further information

Reverts to default mapping of coordinates

Related topics

XADcanvasmap

XADcolor

Purpose

Create a color value.

Synopsis

```
function XADcolor(red:real, green:real, blue:real):integer
```

Arguments

`red` Intensity of red (between 0 and 255)

`green` Intensity of green (between 0 and 255)

`blue` Intensity of blue (between 0 and 255)

Return value

Color value.

Further information

Creates a color value based on intensities of ***red***, ***green*** and ***blue***.

Related topics

See Section 6.3 for a list of predefined color constants.

XADcreatebrowser

Purpose

Create a browser.

Synopsis

```
procedure XADcreatebrowser(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, url:string)
```

Arguments

`wid`	id of the window containing the browser
`id`	Browser identifier
`x`	x coordinate (from left of window) of the browser in pixels
`y`	y coordinate (from top of window) of the browser in pixels
`w`	Width of the browser in pixels
`h`	Height of the browser in pixels
`url`	URL to open when the object is created

Related topics

`XADbrowsergoto`

XADbrowsergoto

Purpose

Open the given URL in the browser.

Synopsis

```
procedure XADbrowsergoto(id:integer, url:string)
```

Arguments

`id`	Browser identifier
`url`	The URL to visit.

Further information

Visits the given ***url***.

Related topics

`XADcreatebrowser`

XADcreatescrollbar

Purpose

Create a scrollbar object.

Synopsis

```
procedure XADcreatescrollbar(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer, vertical:boolean)
```

Arguments

`wid`	id of the window containing the scrollbar
`id`	Scrollbar identifier
`x`	x coordinate (from left of window) of the scrollbar in pixels
`y`	y coordinate (from top of window) of the scrollbar in pixels
`w`	Width of the scrollbar in pixels
`h`	Height of the scrollbar in pixels
`vertical`	***true***=vertical; ***false***=horizontal

Related topics

`XADscrollbarset`, `XADscrollbargetpos`

XADscrollbargetpos

Purpose

Obtain the current position of the scrollbar.

Synopsis

```
function XADscrollbargetpos(id:integer):integer
```

Argument

`id` Scrollbar identifier

Return value

Current scrollbar position.

Related topics

`XADcreatescrollbar`, `XADscrollbarset`

XADscrollbarset

Purpose

Set scrollbar characteristics.

Synopsis

```
procedure XADscrollbarset(id:integer, minimum:integer, maximum:integer,
      pagesize:integer, position:integer)
```

Arguments

`id` Scrollbar identifier

`minimum` Minimum value for scrollbar

`maximum` Maximum value for scrollbar

`pagesize` Size of one "page" (clicking in the scrollbar advances one page at a time)

`position` Initial position of the scrollbar

Related topics

`XADcreatescrollbar`, `XADscrollbargetpos`

XADcreatetree

Purpose

Create a tree object.

Synopsis

```
procedure XADcreatetree(wid:integer, id:integer, x:integer, y:integer,
      w:integer, h:integer)
```

Arguments

`wid`	id of the window containing the tree
`id`	Tree identifier
`x`	x coordinate (from left of window) of the tree in pixels
`y`	y coordinate (from top of window) of the tree in pixels
`w`	Width of the tree in pixels
`h`	Height of the tree in pixels

Related topics

`XADtreeadd`, `XADtreereset`, `XADtreeexpand`

XADtreeadd

Purpose

Add a branch to a tree

Synopsis

```
procedure XADtreeadd(id:integer, parent:string, items:  set of string)
```

Arguments

`id` Tree identifier

`parent` Parent branch identifier (single string or comma-separated strings for deeper items)

`items` Items to show when parent is expanded

Example

```
XADtreeadd(id_tree, "January", {"New York","London","Paris"})
XADtreeadd(id_tree, "February", {"New York","London","Paris"})
XADtreeadd(id_tree, "January,New York", {"Rainfall","Snowfall"})
```

Example result

Further information

Add a branch to the tree. If the ***parent*** is a single string, it is added as a root item. Then all the ***items*** are added as its children. If the ***parent*** is string consisting of comma separated names, XAD will travel down the tree until it finds the corresponding node. Then all the ***items*** are added as its children.

Related topics

`XADcreatetree`, `XADtreereset`, `XADtreeexpand`

XADtreereset

Purpose

Clears the content of a tree

Synopsis

```
procedure XADtreereset(id:integer)
```

Argument

`id` Tree identifier

Related topics

`XADcreatetree`, `XADtreeadd`, `XADtreeexpand`

XADtreeexpand

Purpose

Expands a tree branch .

Synopsis

```
procedure XADtreeexpand(id:integer, parent:string)
```

Arguments

`id` Tree identifier

`parent` Branch identifier (single string or comma-separated strings for deeper items)

Related topics

`XADcreatetree`, `XADtreeadd`, `XADtreereset`

6.19 Events specific to objects

XAD_EVENT_CHANGED	Input, editor or scrollbar changed event.	p. 103
XAD_EVENT_MENU	Menu event.	p. 101
XAD_EVENT_PRESSED	Button pressed event.	p. 103
XAD_EVENT_SELECTION	Selection event.	p. 103
XAD_EVENT_TIMER	Timer event.	p. 101
XAD_EVENT_WINDOW_CLOSED	Window closed event.	p. 101
XAD_EVENT_WINDOW_CLOSING	Window closing event.	p. 102
XAD_EVENT_WINDOW_HIDDEN	Window hidden event.	p. 102
XAD_EVENT_WINDOW_MOVED	Window moved event.	p. 102
XAD_EVENT_WINDOW_OPENED	Window opened event.	p. 102
XAD_EVENT_WINDOW_RESIZED	Window resized event.	p. 102
XAD_EVENT_WINDOW_SHOWN	Window shown event.	p. 102

XAD_EVENT_MENU

Description Menu event.

Note Event generated when a menu item is selected.

XAD_EVENT_TIMER

Description Timer event.

Note Timer event associated with *window* objects. See `XADwindowsettimer`

XAD_EVENT_WINDOW_CLOSED

Description Window closed event.

Note Event generated when a *window* was closed using `XADwindowclose` or by the user.

XAD_EVENT_WINDOW_CLOSING

Description Window closing event.

Note Event generated when a *window* is about to be closed. May be overriden by calling `XADwindowkeep`.

XAD_EVENT_WINDOW_HIDDEN

Description Window hidden event.

Note Event generated when a *window* was hidden using `XADwindowhide` or by the user.

XAD_EVENT_WINDOW_MOVED

Description Window moved event.

Note Event generated when the *window* position has changed.

XAD_EVENT_WINDOW_OPENED

Description Window opened event.

Note Crucial event in the lifetime of a *window* opened using `XADwindowopen`. Perform all object initializations when this event is received.

XAD_EVENT_WINDOW_RESIZED

Description Window resized event.

Note Event generated when a *window* is resized by the user. Capture this event in order to update the positions and sizes of objects in the window, using `XADsetpos`.

XAD_EVENT_WINDOW_SHOWN

Description Window shown event.

Note Event generated when a *window* is shown as a result of calling `XADwindowshow`. Perform all object initializations when this event is received

XAD_EVENT_PRESSED

Description Button pressed event.

Note Event generated by regular *button* objects, *check* buttons, or *radio* buttons.

XAD_EVENT_CHANGED

Description Input, editor or scrollbar changed event.

Note Event indicating that the text in an *input* or *editor* object has changed, or that the position of a *scrollbar* has changed.

XAD_EVENT_SELECTION

Description Selection event.

Note Event indicating that the selection has changed in a *list*, *droplist*, or *tab* object.

Chapter 7

XAD object groups reference

XAD provides functionality to work with groups of objects, quickly allowing multiple objects to be moved, hidden or disabled. This can, for instance, greatly simplify the code required to implement functioning Tab (6.14) objects.

7.1 Group Basics

Unlike the Group (6.7) object which is simply a graphical display, groups are actual collections of linked objects which may be moved, hidden or enabled in unison. An object may belong to multiple groups, but it is important to remember that in these cases the state of the object will depend on the last group command called. For instance if you were to disable a group of objects which consisted of objects A, B and C; then enable a second group containing objects B, D and E, you would find that objects A and C would be disabled, but that B, D and E would all be enabled.

Groups created via `XADgroupcreate` may be referred to directly via the returned group id; however, groups created when loading resources (`XADloadresource`) must first have their ids retrieved using `XADgroupgetid` before operations can be performed upon them.

Specific subroutines

`XADgroupcreate`, `XADgroupaddmember` `XADgroupdisband` `XADgroupremovemember` `XADgroupgetid` `XADgroupgeth` `XADgroupgetw` `XADgroupgetx` `XADgroupgety` `XADgroupsetpos` `XADgroupsetvisible` `XADgroupenable`

Specific events

None

7.2 Subroutines specific to groups

XADgroupcreate

Purpose

Create a group from a list of object ids

Synopsis

```
procedure XADgroupcreate(ids:  set of integer)
```

Argument

`ids` An integer set of object ids for those objects you wish to place in the group

Return value

The group id of the created group.

Related topics

`XADgroupaddmember`, `XADgroupremovemember`, `XADgroupdisband`

XADgroupaddmember

Purpose

Add an object to an already existing group

Synopsis

```
procedure XADgroupaddmember(groupid:  integer, objectid:  integer)
```

Arguments

`groupid` An integer id of the group you wish to add to

`objectid` An integer object id of the object you wish to place in the group

Related topics

`XADgroupcreate`, `XADgroupremovemember`, `XADgroupdisband`

XADgroupdisband

Purpose

Disband/destroy a group of objects

Synopsis

```
procedure XADgroupdisband(id:  integer)
```

Argument

`id` The id of the group you wish to disband.

Related topics

`XADgroupcreate`, `XADgroupaddmember`, `XADgroupremovemember`

XADgroupremovemember

Purpose

Remove an object from a group

Synopsis

```
procedure XADgroupremovemember(groupid:  integer, objectid:  integer)
```

Arguments

`groupid` The group id to remove the object from

`objectid` The id of the object to remove from the group

Related topics

`XADgroupcreate`, `XADgroupaddmember`, `XADgroupdisband`

XADgroupgetid

Purpose

Retrieve the group id by name and window id

Synopsis

```
procedure XADgroupgetid(name:  string, windowid:  integer)
```

Arguments

`name` The name given to the group when creating the resource loaded via `XADloadresource`

`windowid` The id of the window object created when loading the resource containing the group definition

Related topics

`XADloadresource`

XADgroupgeth

Purpose

Find the height of the region enclosing all the objects in the group

Synopsis

```
procedure XADgroupgeth(id:  integer):integer
```

Argument

`id` The id of the group to retrieve the height of

Return value

The height of the group region.

Related topics

`XADgroupcreate`, `XADgroupgetw`, `XADgroupgetx`, `XADgroupgety`

XADgroupgetw

Purpose

Find the width of the region enclosing all the objects in the group

Synopsis

```
procedure XADgroupgetw(id:  integer)
```

Argument

`id` The id of the group to retrieve the width of

Return value

The width of the group region.

Related topics

`XADgroupcreate`, `XADgroupgeth`, `XADgroupgetx`, `XADgroupgety`

XADgroupgetx

Purpose

Find the x-position of the start of the region enclosing all the objects in the group

Synopsis

```
procedure XADgroupgetx(id:  integer):integer
```

Argument

`id` The id of the group to retrieve the x-position

Return value

The x-position of the group region.

Related topics

`XADgroupcreate`, `XADgroupgeth`, `XADgroupgetw`, `XADgroupgety`

XADgroupgety

Purpose

Find the y-position of the start of the region enclosing all the objects in the group

Synopsis

```
procedure XADgroupgety(id:  integer):integer
```

Argument

`id` The id of the group to retrieve the y-position

Return value

The y-position of the group region.

Related topics

`XADgroupcreate`, `XADgroupgeth`, `XADgroupgetw`, `XADgroupgetx`

XADgroupsetpos

Purpose

Move the group objects to a new position (keeping all objects in the group at the same relative positions)

Synopsis

```
procedure XADgroupsetpos(groupid:  integer, xpos:  integer,
       ypos:integer):integer
```

Arguments

`groupid` The id of the group to move

`xpos` The x-position to move the group region to

`ypos` The y-position to move the group region to

Related topics

`XADgroupcreate`, `XADgroupgetx`, `XADgroupgety`

XADgroupsetvisble

Purpose

Hide or show the objects within the group

Synopsis

```
procedure XADgroupsetvisible(id:  integer, show:  boolean)
```

Arguments

`id`	The id of the group to show/hide
`show`	If true the objects within the group are shown, if false they are hidden

Related topics

`XADgroupcreate`, `XADgroupenable`, `XADgroupsetpos`

XADgroupenable

Purpose

Enable or disable the objects within the group

Synopsis

```
procedure XADgroupenable(id:  integer, enable:  boolean)
```

Arguments

`id` The id of the group to enable/disable

`enable` If true the objects within the group are enabled, if false they are disabled

Related topics

`XADgroupcreate`, `XADgroupsetvisible`, `XADgroupsetpos`

Chapter 8

Generic routines

XADdestroy

Purpose

Delete an object.

Synopsis

```
procedure XADdestroy(id:integer)
```

Argument

`id` Object identifier

Further information

When an object is no longer needed, it can be destroyed. Once destroyed, its ***id*** can no longer be used. However, a new object can be created with the same ***id***.

XADenable

Purpose

Enable/disable user interaction.

Synopsis

```
procedure XADenable(id:integer, state:boolean)
```

Arguments

`id` Object identifier

`state`

- `true` enable user interaction with the object
- `false` disable user interaction with the object

Further information

Used to enable/disable user interaction with the object. For example, a *Save* button should probably be disabled if there is nothing to save.

XADgetmousex

Purpose

Get the x coordinate of the mouse cursor relative to an object.

Synopsis

```
function XADgetmousex(id:integer):integer
```

Argument

`id` Object identifier

Return value

x coordinate of the mouse cursor, relative to the object.

Further information

This function returns the x coordinate (in pixels) of the mouse cursor, relative to the upper left corner of the object specified.

Related topics

`XADgetmousey`

XADgetmousey

Purpose

Get the y coordinate of the mouse cursor relative to an object.

Synopsis

```
function XADgetmousey(id:integer):integer
```

Argument

`id` Object identifier

Return value

y coordinate of the mouse cursor, relative to the object.

Further information

This function returns the y coordinate (in pixels) of the mouse cursor, relative to the upper left corner of the object specified.

Related topics

`XADgetmousex`

XADgetx

Purpose

Get the x coordinate of an object.

Synopsis

```
function XADgetx(id:integer):integer
```

Argument

`id` Object identifier

Return value

x coordinate value of the object.

Further information

This function returns the x coordinate (in pixels) of the object ***id*** relative to the upper left corner of the window containing it. If ***id*** refers to a *window*, it returns the coordinate relative to the screen.

Related topics

`XADgety`, `XADgeth`, `XADgetw`

XADgety

Purpose

Get the y coordinate of an object.

Synopsis

```
function XADgety(id:integer):integer
```

Argument

`id` Object identifier

Return value

y coordinate value of the object.

Further information

This function returns the y coordinate (in pixels) of the object ***id*** relative to the upper left corner of the window containing it. If ***id*** refers to a *window*, it returns the coordinate relative to the screen.

Related topics

`XADgetx`, `XADgeth`, `XADgetw`

XADgetw

Purpose

Get the width of an object.

Synopsis

```
function XADgetw(id:integer):integer
```

Argument

`id` Object identifier

Return value

Width of the object.

Further information

This function returns the width (in pixels) of the object. If the identifier ***id*** represents a window, and its value is negative (e.g. -1000000000), then the **inner** width of the window is returned. This represents the 'useable' space of a window, as it excludes borders, menus, titles, etc.

Related topics

`XADgeth`, `XADgetx`, `XADgety`

XADgeth

Purpose

Get the height of an object.

Synopsis

```
function XADgeth(id:integer):integer
```

Argument

`id` Object identifier

Return value

Height of the object.

Further information

This function returns the height (in pixels) of the object. If the identifier ***id*** represents a window, and its value is negative (e.g. -1000000000), then the **inner** height of the window is returned. This represents the 'useable' space of a window, as it excludes borders, menus, titles, etc.

Related topics

`XADgetw`, `XADgetx`, `XADgety`

XADgetid

Purpose

Get the id of an object.

Synopsis

```
function XADgetid(name:string, windowid:integer):integer
```

Arguments

`name` Object name as set in the resource file, or via `XADsetname`.

`windowid` The id of the window the object belongs to.

Return value

The id of the object.

Further information

Related topics

`XADloadresource`, `XADgetw`, `XADgetx`, `XADgety`

XADloadresource

Purpose

Load a resource file as a XAD window.

Synopsis

```
function XADloadresource(file: string):integer
```

Argument

`file` The name of the resource file to load

Return value

The window id of the newly created XAD window object.

Further information

The resource file is an XML-like file created by the IVE XAD editor and is used to quickly create a XAD window and associated objects. The various ids for the objects may be retrieved using `XADgetid`.

Related topics

`XADgetid`, `XADgroupgetid`, `XADsetname`

XADrefresh

Purpose

Refresh an object.

Synopsis

```
procedure XADrefresh(id:integer)
```

Argument

`id` Object identifier

Further information

If an object may not be painted correctly due to a complex layout operation, use this routine to update its appearance.

XADsetfocus

Purpose

Focus on an object

Synopsis

```
procedure XADsetfocus(id:integer)
```

Argument

`id` Object identifier

Further information

The target object will receive the keyboard focus.

XADsetname

Purpose

Set or change the name of an object

Synopsis

```
procedure XADsetname(id:integer, name:  string)
```

Arguments

`id`	Object identifier
`name`	The name to give the object

Further information

Objects loaded via a resource will have names set from the resource file. Objects constructed via code will not have names and so you can set them with this routine if you desire (to allow use of the non-generic event callbacks normally generated by the IVE XAD editor event dialog). Be careful if altering the names of objects created and loaded via a resource file; any event callbacks created by the IVE XAD editor event dialog will not be called if the names of the objects are changed.

Related topics

`XADloadresource`

XADsetpos

Purpose

Reposition an object.

Synopsis

```
procedure XADsetpos(id:integer, x:integer, y:integer, w:integer, h:integer)
```

Arguments

`id`	Object identifier
`x`	x coordinate
`y`	w coordinate
`w`	Width
`h`	Height

Further information

This procedure repositions the object ***id***. Note: use the `XADget*` routines to leave parameters unchanged.

Related topics

`XADgeth`, `XADgetw`, `XADgetx`, `XADgety`

XADsettext

Purpose

(Re)set the textual information of an object.

Synopsis

```
procedure XADsettext(id:integer, text:string)
```

Arguments

`id`	Object identifier
`text`	New textual information

Further information

This procedure updates the textual information of an object if applicable.

XADsetvisible

Purpose

Making an object visible/hidden.

Synopsis

```
procedure XADsetvisible(id:integer, visible:boolean)
```

Arguments

`id` Object identifier

`visible` Display option.

- `true` object visible
- `false` object hidden

Further information

Objects may be shown or hidden by calling this procedure.

XADgeteventtext

Purpose

Retrieve the message associated with an event.

Synopsis

```
function XADgeteventtext:string
```

Return value

The event message or an empty string.

Further information

If an event carries textual information (for example a tab selection), use this function to retrieve the text when handling the event.

Related topics

XAD_EVENT_CHANGED, XAD_EVENT_SELECTION, XAD_EVENT_KEYDOWN, XAD_EVENT_KEYUP

Chapter 9

Generic events

XAD_EVENT_KEYDOWN

Description Key pressed.

Note Indicates that a key was *pressed* when the object had the focus. Call `XADgeteventtext` to obtain the representation of the key that was pressed. Letters and digits are returned as themselves. The following special codes can also be returned:

Up arrow	`"up"`
Down arrow	`"down"`
Left arrow	`"left"`
Right arrow	`"right"`
Tab	`"tab"`
Enter	`"enter"`
Shift	`"shift"`
Ctrl	`"control"`
Caps Lock	`"capslock"`
Esc	`"esc"`
Spacebar	`" "`
Page Up	`"pageup"`
Page Down	`"pagedown"`
End	`"end"`
Home	`"home"`
Insert	`"ins"`
Delete	`"del"`

XAD_EVENT_KEYUP

Description Key released.

Note Indicates that a key was *released* when the object had the focus. See `XAD_EVENT_KEYDOWN` for more information.

XAD_EVENT_MOUSE_LEFTDBCLK

Description Double click with left mouse button.

Note Indicates that the left mouse button was *double clicked* above the object. When a user double clicks, the following messages are generated by Windows, in this exact sequence:

```
XAD_EVENT_MOUSE_LEFTDOWN
XAD_EVENT_MOUSE_LEFTUP
XAD_EVENT_MOUSE_LEFTDBLCLK
XAD_EVENT_MOUSE_LEFTUP
```

XAD_EVENT_MOUSE_LEFTDOWN

Description Left mouse button pressed.

Notes Indicates that the left mouse button was *pressed* above the object. The coordinates of the mouse relative to ANY object ***id*** can be obtained ANYTIME using `XADgetmousex` and `XADgetmousey`.
When double clicking on an object the second click is interpreted by Windows as 'left button double click'.

See also `XAD_EVENT_MOUSE_LEFTDBLCLK`

XAD_EVENT_MOUSE_LEFTUP

Description Left mouse button released.

Note Indicates that the left mouse button was *released* above the object.

See also `XAD_EVENT_MOUSE_LEFTDOWN`, `XAD_EVENT_MOUSE_LEFTDBLCLK`

XAD_EVENT_MOUSE_MOVED

Description Mouse moved.

Note Indicates that the mouse has *moved* above the object.

XAD_EVENT_MOUSE_RIGHTDBCLK

Description Double click with right mouse button.

Note Indicates that the right mouse button was *double clicked* above the object. When a user double clicks, the following messages are generated by Windows, in this exact sequence:

```
XAD_EVENT_MOUSE_RIGHTDOWN
XAD_EVENT_MOUSE_RIGHTUP
XAD_EVENT_MOUSE_RIGHTDBLCLK
XAD_EVENT_MOUSE_RIGHTUP
```

XAD_EVENT_MOUSE_RIGHTDOWN

Description Right mouse button pressed.

Note Indicates that the right mouse button was *pressed* above the object. When double clicking on an object the second click is interpreted by Windows as 'right button double click'.

See also `XAD_EVENT_MOUSE_RIGHTDBLCLK`

XAD_EVENT_MOUSE_RIGHTUP

Description Right mouse button released.

Note Indicates that the right mouse button was *released* above the object.

See also `XAD_EVENT_MOUSE_RIGHTDBLCLK`, `XAD_EVENT_MOUSE_RIGHTDOWN`

Chapter 10

Utility routines

XADid

Purpose

Obtain a unique identifier to represent objects.

Synopsis

```
function XADid:integer
```

Return value

Automatically incremented unique identifier.

Example

```
declarations
  id_win=XADid
  id_canvas=XADid
  id_button=XADid
  ...
end-declarations
```

XADsavescreenshot

Purpose

Take a screenshot of an object and save it to a file.

Synopsis

```
procedure XADsavescreenshot(id:integer, filename:string)
```

Arguments

`id` Object identifier (may be a window or any other object type)

`filename` Name of image file (must have one of these extensions: .jpg, .gif, .bmp, .png)

Example

```
XADsavescreenshot(id_canvas,"canvas.png")
XADsavescreenshot(id_win,"MyXADApplication.jpg")
```

XADseteventcallback

Purpose

Set the event handler callback

Synopsis

```
procedure XADseteventcallback(handlername:string)
```

Argument

`handlername` Event handler callback

Further information

This procedure registers the procedure ***handlername*** to act as an event handler callback. All events will be reported through this callback. The callback procedure has this signature:

```
procedure guievents(id:integer, event:integer)
```

and it gets called for every possible event. The user has the option to ignore or deal with events through the event handler callback procedure.

XADhandleevents

Purpose

Handling events during program execution.

Synopsis

```
procedure XADhandleevents
```

Further information

If a long calculation (*e.g.* optimization) is initiated by the event handler callback, the user interface will freeze. This is due to the fact that the processor intensive operations take place on the same thread as the code that draws the user interface or responds to user events (after all, we are dealing with an event). To avoid this phenomenon, call `XADhandleevents` at regular intervals (*e.g.* during Optimizer callbacks) to allow the user to interact with the user interface. Use caution, however as the following call sequence is likely to occur:

```
        guievents(1,1)
 calls     minimize(objective)
 calls        globallog
 calls           XADhandleevents
 calls              guievents(1,10)
 calls                 ???
```

While an optimization is running events should be dealt with quickly and with little (if well understood) or no side effects. In this situation one could use `XADseteventcallback` to switch to an alternative, simplified event handler callback.

Related topics

`XADseteventcallback`

XADchoosefile

Purpose

Display file selection dialog.

Synopsis

```
function XADchoosefile(openorsave:boolean):string
function XADchoosefile(openorsave:boolean, filetypes:string):string
function XADchoosefile(openorsave:boolean, filetypes:string,
      dir:string):string
```

Arguments

`openorsave` Dialog type selection.

`true`	create an *Open file* dialog
`false`	create a *Save file* dialog

`filetypes` File filters based on file extensions. See example below

`dir` The initial folder displayed in the file dialog.

Return value

File name if selection was successful, otherwise an empty string.

Example

```
filename:=XADchoosefile(true,"My own type of files (*.myotf)|*.myotf"+
                            "|"+
                            "MPS files(*.mps)|*.mps"+
                            "|"+
                            "All Files (*.*)|*.*"+
                            "|")
```

Further information

This is a convenience routine for displaying the standard Windows file selection dialog. Pass `true` as an argument to create an *Open file* dialog and `false` to create a *Save file* dialog. If the file selection was successful, the returned string contains the file name. If not, the function returns an empty string. The second form of the function allows use of filters based on the file extension.

XADpopupmenu

Purpose

Creation of a pop-up menu.

Synopsis

```
function XADpopupmenu(menuitems:set of string):string
```

Argument

`menuitems` Menu items

Return value

Selected item, or empty string if no selection was made.

Example

To create a menu with the items *Action A* and *Action B* separated by a line, use the following:

```
choice:=XADpopupmenu({"ActionA", "XADseparator", "Action B"})
```

Further information

It is customary for user interfaces to display a menu of options when the user right-clicks on something. This routine achieves just that. When the user makes a choice, it is returned as a string. If the user does not select anything, the returned string will be empty.

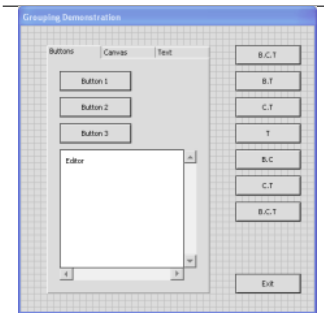
Chapter 11

XAD Examples

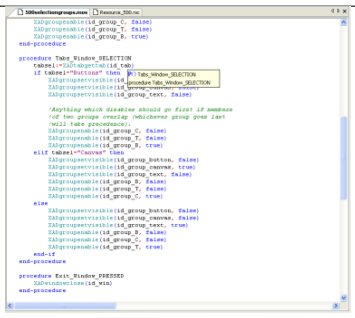
11.1 Resource Example

The following tutorial is taken from the IVE help documentation. If you have not used IVE previously and wish to see a more in-depth explanation of the various dialog controls it is recommended that you view the IVE compiled help.

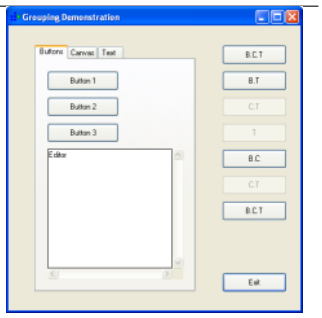
In order to demonstrate the use of the XAD Resource Editor and the associated XAD Mosel commands we will now look at the example "500selectiongroups.mos", in the XAD examples folder of the Xpress installation. This example not only covers the use of resources, but the manipulation of resource generated groups within Mosel code and the use of an object in multiple groups.



+



=>



To begin with load the "500selectiongroups.mos" file in to IVE and view the behaviour of the model when run.

In the example the tab object works by picking up the tab's *SELECTION* event, calling the relevant Mosel callback (*Tabs_Window_SELECTION*) and then setting the enabled and visible flags of the objects relevant to the currently selected tab. In this case the code required to do this is reasonably simple as we have setup groups of objects which we may hide or show with one command. Herein lies the power of object groups.

We will now look at the various sections of the example's Mosel code, before looking at the associated resource in the XAD Resource Editor.

11.1.1 The Mosel Code

```
500selectiongroups.mos | Resource_500.rsc

    XADgroupenable(id_group_C, false)
    XADgroupenable(id_group_T, false)
    XADgroupenable(id_group_B, true)
end-procedure

procedure Tabs_Window_SELECTION
    tabsel:=XADtabgettab(id_tab)
    if tabsel="Buttons" then
        XADgroupsetvisible(id_
        XADgroupsetvisible(id_
        XADgroupsetvisible(id_group_text, false)

        !Anything which disables should go first if members
        !of two groups overlap (whichever group goes last
        !will take precedence).
        XADgroupenable(id_group_C, false)
        XADgroupenable(id_group_T, false)
        XADgroupenable(id_group_B, true)
    elif tabsel="Canvas" then
        XADgroupsetvisible(id_group_button, false)
        XADgroupsetvisible(id_group_canvas, true)
        XADgroupsetvisible(id_group_text, false)
        XADgroupenable(id_group_B, false)
        XADgroupenable(id_group_T, false)
        XADgroupenable(id_group_C, true)
    else
        XADgroupsetvisible(id_group_button, false)
        XADgroupsetvisible(id_group_canvas, false)
        XADgroupsetvisible(id_group_text, true)
        XADgroupenable(id_group_B, false)
        XADgroupenable(id_group_C, false)
        XADgroupenable(id_group_T, true)
    end-if
end-procedure

procedure Exit_Window_PRESSED
    XADwindowclose(id_win)
end-procedure
```

P() Tabs_Window_SELECTION
procedure Tabs_Window_SELECTION

The Mosel code has the following parts (ignoring those parts common to standard Mosel models):

1. **Load the window from resource:** Here we load the resource file in to the model. All resources equate to one XAD window and the return value of the function used to load the resource, *XADloadresource*, is the *id* of the XAD window object (*id_win*, in this case).

2. **Retrieve the object/group ids:** Although we need not retrieve the object ids for all of the resource objects, if we wish to manipulate or respond to events for that object we must do so. When creating the resource each object/group will have been given a name (either the default "XAD_*OBJECTTYPE_COUNTER*", or set by the user) and it is this that we will use to retrieve the object ids.

   Using the XAD functions *XADgetid* and *XADgroupgetid* we may retrieve the ids for objects and groups, respectively.

3. **Display the window:** This function opens the specified XAD window and displays all the associated objects. In order to only display/enable those objects relevant for the initially displayed tab we will need to setup the object states when the window opens. This is achieved via a *WINDOW_OPENED* event callback.

4. ***procedure Window_WINDOW_OPENED*:** In this callback we need to setup the various states of the objects/groups belonging to each tab selection. There are six groups within the example, 3 relating directly to those objects displayed on each tab, and 3 relating to the buttons on the right hand side of the example.

The right hand side buttons demonstrate that when the **B**utton, **C**anvas or **T**ext tabs are selected the relevant buttons are enabled or disabled. These differ from the groups setup for the objects in the tab control as each button may belong to more than one group (Group **B**, **C** or **T** depending on which tab selections they will be enabled for).

We intially have the "*Button*" tab selected and so within this callback we enable the *id_group_button* and *id_group_B* groups and disable the others.

Note: there are no events associated with the buttons in this example and so they will not actually perform any action if clicked.

5. ***procedure Tabs_Window_SELECTION*:** This callback is in essence very similar to the *WINDOW_OPENED* callback, above. The difference being that we must check for the currently selected tab and then disable/enable and show/hide the relevant groups for each tab.

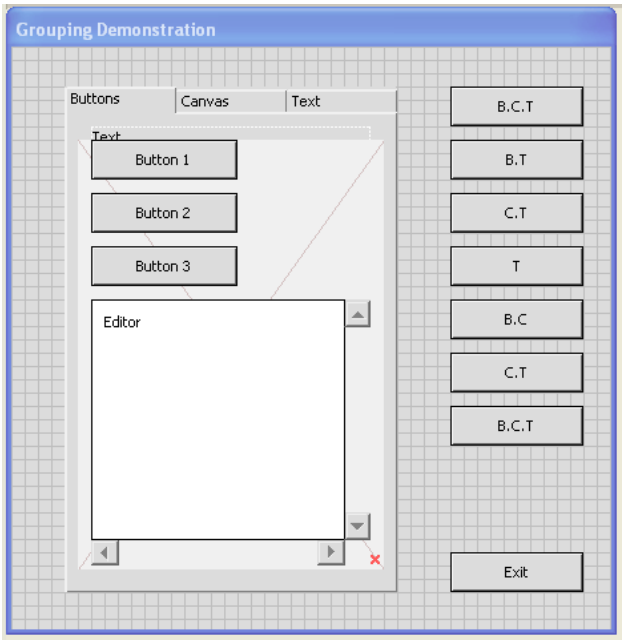
   As the comment in the code mentions, it is important to get the order of the commands correct if you are dealing with objects in multiple groups. Were the command order incorrect you may inadvertently enable and then disable an object (belonging to multiple groups) that you intended to be enabled.

   Note: It is recommended to first disable all the groups you need to before finally enabling the relevant group or groups (as in this example).

6. ***procedure Exit_Window_PRESSED*:** When the "Exit" button is *PRESSED* this callback is called. All it does is cleanly close the *id_win* window so that the program closes in a user controlled and clean manner.

11.1.2 The Associated Resource File

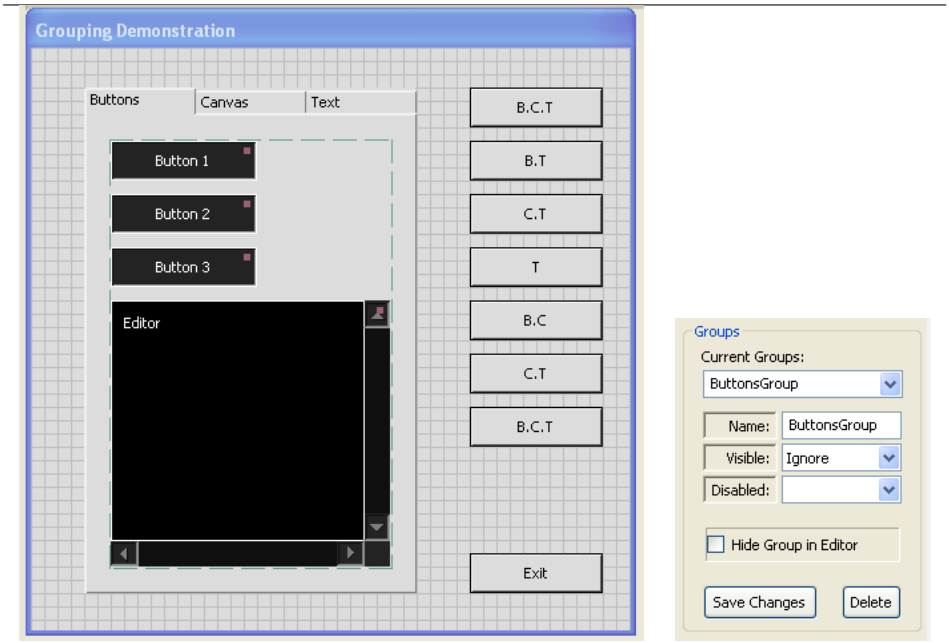
The resource file associated with the example, "Resource_500.rsc", can be found in the XAD examples directory alongside the Mosel file. Once loaded in to IVE you will be presented with the XAD Resource Editor and the representation of the XAD window and objects will be visible in the Form Edit Dialog (FED).



When initially loaded all of the objects within the tab will be visible. To hide a group of objects within the editor select the group from the XAD Properties Dialog Group drop-down list and then select the option to *Hide Group in Editor*.

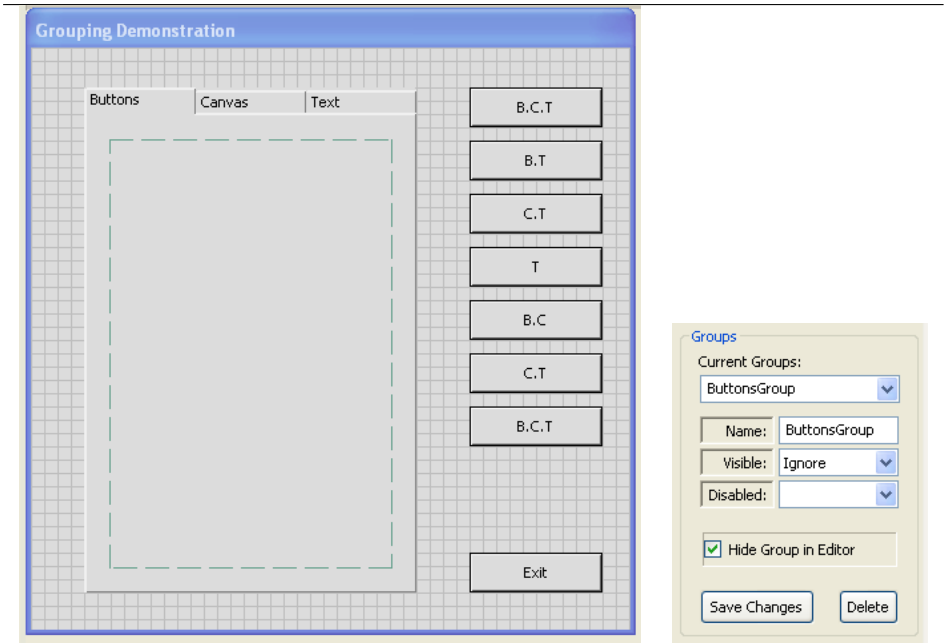
This behaviour can be used to quickly shift between group selections designed for use in tab objects. In this example we will hide the *Buttons* group and show the *Canvas* group:

1. **Select the group you wish to hide:**



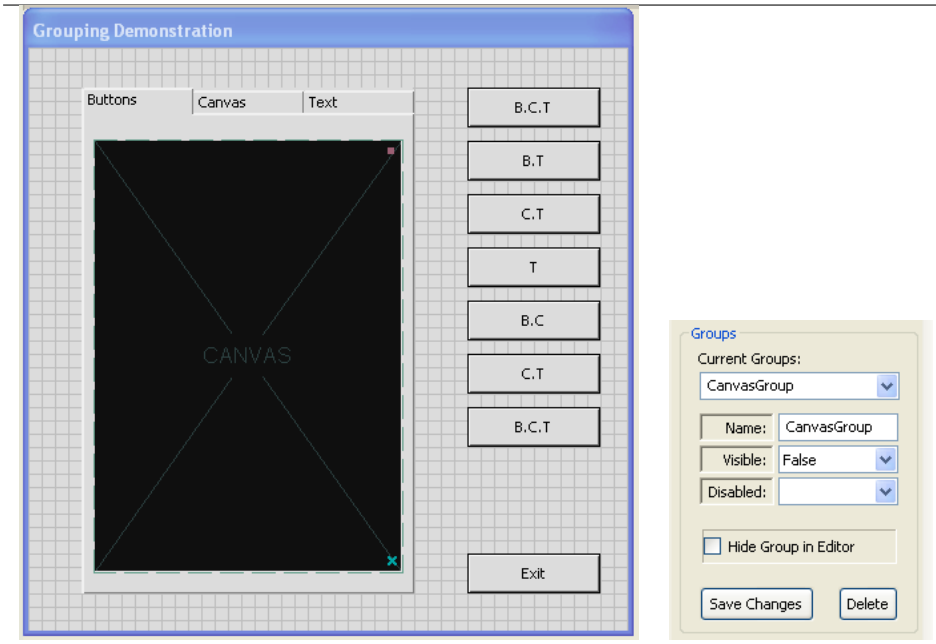
Here we've selected the *Buttons* group. It's currently visible in the editor.

2. **Hide the *Buttons* group:**



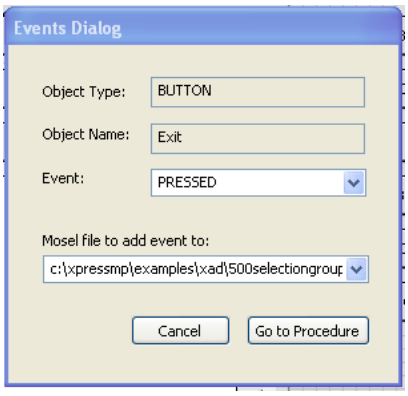
By selecting the *Hide Group in Editor* option we hide the *Buttons* group.

3. **Unhide the *Canvas* group:**



By selecting the *Canvas* group in the drop-down list, and unselecting the *Hide Group in Editor* option, the *Canvas* group becomes visible in the editor.

The events for the objects on the resource may be added, or navigated to, via the XAD Event Dialog. This is shown when an object, or the form itself, is double left-clicked.



As an example we will now navigate to the "Exit" button callback discussed earlier. To do so, firstly double click the "Exit" button in the FED to open the Event Dialog for the button. Once this is open we can navigate to the event callback in the following manner:

1. **Select the Event:** In this case we wish to select the *PRESSED* event, but were you adding a different event you could select any of the events offered to you in the drop-down list.
2. **Select the Mosel File:** We wish to navigate to the event callback already set in "500selectiongroups.mos", but you could choose to add the event to any valid Mosel file in which you intend to load the "Resource_500.rsc".
3. **Go to Procedure:** Once you've selected the event and file you wish click the button and you will be taken to the relevant callback in the file specified. In this case the callback already exists and so you should now see the *XADwindowclose* function call which forms the operational code of the callback.

If you'd chosen to add a currently non-existent event callback to the file then the code part of the callback would contain the default "*Not yet implemented*" Mosel text output.

11.2 Non-Resource Example

A simple assignment problem will be used to illustrate how XAD works with Mosel to create interactive mathematical programming models.

Here is a screenshot of the application:

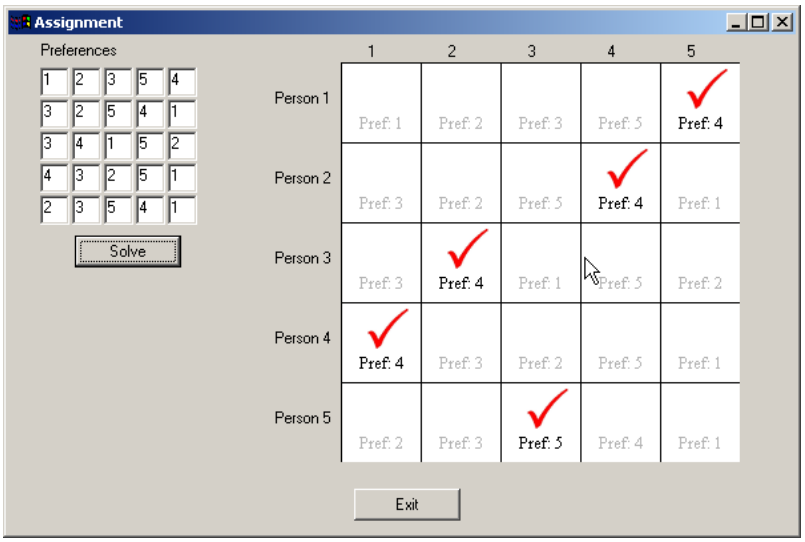


Figure 11.1: Interactive mathematical programming model

The user can modify the preferences by changing the numbers in the input boxes in the upper left corner. After pressing *Solve*, the results are shown in the diagram. This procedure can be repeated any number of times.

Let's examine the code, piece by piece:

```
model Assignment
 uses "mmxprs","mmxad"

 declarations
  NP = 5                                  ! Number of persons/projects
  RP = 1..NP                              ! Set (range) of persons/projects
  PREF: array(RP,RP) of integer           ! Preference values
  assign: array(RP,RP) of mpvar           ! Assignment person-project
 end-declarations
```

Here, we tell Mosel that we need to use Xpress-Optimizer (*mmxprs*) as well as XAD (*mmxad*) to build this application. We also declare some identifiers to be used later in the model, such as the decision variable array ***assign***.

In the next section:

```
 PREF:= [1, 2, 3, 5, 4,
         3, 2, 5, 4, 1,
         3, 4, 1, 5, 2,
         4, 3, 2, 5, 1,
         2, 3, 5, 4, 1]
! Objective function: maximize satisfaction
 Satisfaction:= sum(m,p in RP) PREF(m,p)*assign(m,p)
! One person per project
 forall(p in RP) OnePersProj(p):= sum(m in RP) assign(m,p)=1
! One project per person
 forall(m in RP) OneProjPers(m):= sum(p in RP) assign(m,p)=1
```


We assign some suggested preferences (these will be editable by the user). The objective function *Satisfaction* reflects the overall sum of preferences depending on whether assignments are made or not. Two sets of constraints, one person per project and one project per person complete the mathematical formulation of this model.

Let's examine the user interface code:

```
declarations
    id_win=1;       id_textprefs=2;     id_buttonsolve=3;
    id_canvas=4;    id_buttonexit=5;    id_inputs=10; id_texts=100;
end-declarations

 XADcreatewindow    (id_win,100,100,600,400,                    "Assignment")
    XADcreatetext   (id_win,id_textprefs,24,4,65,15,            "Preferences")
    forall(i in 1..NP,j in 1..NP) do
        XADcreateinput(id_win,id_inputs+10*i+j,24*j,24*i,22,22, ""+PREF(i,j))
    end-do
    XADcreatebutton (id_win,id_buttonsolve,50,150,80,24,        "Solve")
    forall(i in 1..NP) do
        XADcreatetext(id_win,id_texts+(NP+i),270+(i-1)*60,5,15,15,""+i)
        XADcreatetext(id_win,id_texts+i,200,40+(i-1)*60,45,15,    "Person "+i)
    end-do
    XADcreatecanvas(id_win,id_canvas,250,20,300,300)
    XADcreatebutton(id_win,id_buttonexit,260,340,80,24,         "Exit")
```

In the code above, a new group of declarations is used to assign unique ids to various GUI objects. The window is created first, then the XAD objects are created one by one, with `id_win` as their *parent*. Note that `id_inputs` and `id_texts` are special in the sense that they are used in combination with `i` and `j` to create unique ids for more than one item. Also note how the expresion `""+PREF(i,j)` actually fills each input object with the corresponding preference rating.

The coordinates and dimensions of all the GUI objects can be derived using the Xpress Application Developer Designer tool, which is a Mosel program (written with XAD) that acts as a *What You See Is What You Get* GUI editor. Some experimentation with layout may be needed before the end result is satisfactory.

The use of *integers* as ids for XAD objects facilitates grouping objects in easy to understand and manage categories (such as all the *inputs* above). The user should take advantage of this feature, especially for large models/applications.

The canvas `id_canvas` displays the results of our optimization problem. An entire procedure is dedicated to updating this object with the most recent information:

```
procedure UpdateCanvas
    XADcanvaserase(id_canvas,XAD_WHITE)
    !draw the assignment table
    forall(i in 1..NP) do
        XADcanvasdrawline(id_canvas,(i-1)*60,0,(i-1)*60,300,XAD_BLACK)
        XADcanvasdrawline(id_canvas,0,(i-1)*60,300,(i-1)*60,XAD_BLACK)
        forall(j in 1..NP) do
            if getsol(assign(i,j))<>0 then
                XADcanvasdrawimage  (id_canvas,(j-1)*60+20,(i-1)*60+5,31,30,
                                        "checkmark.bmp")
                XADcanvasdrawtext   (id_canvas,(j-1)*60+30,(i-1)*60+45,
                                        "Pref: "+PREF(i,j), XAD_BLACK)
            else
                XADcanvasdrawtext   (id_canvas,(j-1)*60+30,(i-1)*60+45,
                                        "Pref: "+PREF(i,j), XADcolor(180,180,180))
            end-if
        end-do
    end-do
    XADcanvasrefresh(id_canvas)
end-procedure
```

As a general rule, a canvas should be erased first using `XADcanvaserase`. After all the drawing is complete, call `XADcanvasrefresh` to update the contents of the canvas.

The procedure `UpdateCanvas` draws a grid and then updates each cell based on the optimized values in the `assign` array. If the assignment is made, a check mark is also drawn in the cell from a bitmap image file. If an assignment is not made, the text in the cell is drawn with a lighter shade of gray to de-emphasize it.

We shall now examine the event handling callback procedure and the code that kick starts the application.

```
62 procedure guievents(id:integer, event:integer)
63     if id=id_buttonsolve and event=XAD_EVENT_PRESSED then
64         !update preferences
65             forall(i in 1..NP,j in 1..NP) do
66                 PREF(i,j):=integer(XADinputgettext(id_inputs+10*i+j))
67             end-do
68         !update objective function
69             Satisfaction:= sum(m,p in RP) PREF(m,p)*assign(m,p)
70             maximize(Satisfaction)
71         !update canvas
72             UpdateCanvas
73     elif id=id_buttonexit and event=XAD_EVENT_PRESSED then
74         XADwindowclose(id_win)
75     elif id=id_win and event=XAD_EVENT_WINDOW_OPENED then
76         UpdateCanvas
77     end-if
78 end-procedure
79
80 XADseteventcallback("guievents")
81 XADwindowopen(id_win)
82
83 end-model
```

Three events are of interest to us in this application: When the window opens and when either of the two buttons is pressed.

- When the `id_buttonsolve` button is pressed Mosel must update the array of preferences (taken directly from the *input* objects), update the objective function based on the new preferences, optimize the problem, and finally update the canvas so that it shows the new set of assignments.
- When the `id_buttonexit` button is pressed, the window is closed immediately.
- The event `XAD_EVENT_WINDOW_OPENED` should be handled to update the status of a window before the user has a chance to interact with it. It is the first event in the lifetime of a window.

Two more statements in the Mosel code are of interest. We must ensure that `XADseteventcallback` is called *before* opening the window, so that the window can send its events to it. Finally, `XADwindowopen` opens the window, giving it control over the execution (through events).

Note that when we call `XADwindowclose` or when we close the window with the mouse, execution of the Mosel code in fact continues with the statement following `XADwindowopen` (in this case there's nothing else to execute, so the application ends). This means that we should always think of windows as mere components of a Mosel application that temporarily gain control of the Mosel execution through the event handler. Mosel is always in charge and can dismiss a window at any time.

Index

Y