CONTENTS

Creating Dialog boxes from the MUI Xtra

Overview

The MUI Xtra provides fully functional dialog boxes set up the way that you specify. These dialog boxes don't require the memory or disk footprint of a MIAW that simulates a dialog box. They also draw the appropriate Windows controls, follow preferences for dialog box appearance set by the user in Windows 95, and draw with gray-scale controls on the Macintosh.

Dialog boxes created from the MUI Xtra also provide:

- Horizontal scrolling within editable text fields
- Clipboard operations for editable text
- Activation of a default item when the user presses Enter or Return
- Activation of a cancel item when the user presses Esc-period or
- Command-period.

The MUI Xtra supports these dialog box types:

- General purpose dialog boxes that you can set up with any arrangement of buttons, editable fields, labels, sliders, and pop-ups
- Alerts that offer several possible button and icon combinations
- Standard open file dialog boxes
- Standard save file dialog boxes
- A dialog box for entering a URL

See Creating dialog boxes [xref] for an explanation of how to create dialog boxes.

What the MUI Xtra contains

The MUI Xtra contains several components that you can use to create dialog boxes:

- These Lingo elements:

  Alert
  FileOpen
  FileSave
  GetItemPropList
  GetUrl
  GetWidgetList
  GetWindowPropList
  Initialize
  ItemUpdate
  New
  Run
  Stop
  WindowOperation
- A predefined list of window properties that you can use as default values for a general purpose dialog box
- A predefined list of properties for components of a general purpose dialog box

Creating dialog boxes

To create dialog boxes from the MUI Xtra:

1. Plan and design the dialog box.

   Determine what users need the dialog box to achieve, the type of interface components that best accomplish this, and where the data in the dialog box comes from. Sketch a preliminary dialog box to get a better idea of where options need to appear and how well they fit into the dialog box.
2. Decide which type of dialog box to use.

   The MUI Xtra provides predefined dialog boxes for entering alerts, opening and saving files, and for entering URLs. These are relatively easy to create. If none of these types are appropriate, you can define your own general purpose dialog box.
3. Use the statement new (xtra "MUI") to create an instance of the MUI Xtra.
4. Set up the dialog box.

You can create two types of dialog boxes:

- Standard dialog boxes that you use for alerts, opening and saving files, and opening URLs. When the user clicks a button, the dialog box responds accordingly and sends the appropriate message back to Director.

  Create these by calling the function that creates the specific dialog box.

- General purpose dialog boxes for which you define each attribute and component in the dialog box, and then explicitly initialize and run the dialog box. See Using a general purpose dialog box for more information.

Creating a standard dialog box

Create and customize a standard dialog box by using the function for the specific type of box.

To create an alert:

Use the Alert function.

To create an open file dialog box:

Use the FileOpen function.

To create a save file dialog box:

Use the FileSave function.

To create a dialog box for entering a URL:

Use the GetUrl function.

Using a general purpose dialog box

When you use a general purpose dialog box, first set up the dialog box's overall attributes and components. After the dialog box's attributes are set up, use the Initialize, Run, and Stop commands to control the dialog box.

To define a general purpose dialog box:

1 Create two lists of definitions:

- ► One list is a property list that contains definitions of overall dialog box attributes such as the name that appears in the title bar, the box's size and location, and the handler that runs when an event occurs in the dialog box. See Specifying overall dialog box properties
- ► The other list is a linear list of definitions for each component in the dialog box. Each item in this linear list is a property list that defines one component of the dialog box. See Specifying dialog box content. [xref]

2 Use the Initialize command to specify that these lists are the ones Director should use for definitions of the general purpose dialog box.

A Lingo statement can't contain more than 256 characters. Since lists are typically long, it might be necessary to assign some values to variables and then use the variables in the list.

To use a general purpose dialog box:

After the dialog box is defined and initialized, use the appropriate Lingo to open, respond to, and close the dialog box.

- ► Use the Run command to open a modal dialog box. Use the WindowOperation command with the #show option to open a non-modal dialog box.
- ► Use the ItemUpdate command to update the dialog box in response to user actions.
- ► Respond to user actions by sending events in the dialog box to the handler set up to handle callbacks. See Sending dialog box events to Director [xref].
- ► Use the Stop command to close a modal dialog box. Use the WindowOperation command with the #hide option to close a non-modal dialog box.

Specifying overall dialog box properties

To specify the dialog box's overall properties, create a property list that sets values for each of the window's properties.

The GetWindowPropList function returns a predefined list of attributes. It's usually easiest to obtain a predefined list from the GetItemPropList function and then modify values as needed.

Before opening the dialog box, use the Initialize command to specify which list Director uses as the source of definitions for the overall dialog box.

The following are the overall dialog box properties and their possible values:

Property	Possible Values
#type	#alert, #normal, #palette
#name	String that contains the window name. Use "" for no name.
#callback	Handler that processes the result of the callback. See Sending dialog box events to Director for more information.
#mode	#data, #dialogUnit, or #pixel. These set the way that Director lays out the dialog box.
#Xposition	Number of pixels that upper left corner of the dialog box appears from the left of the screen. Specify -1 to have the dialog box appear in the center.
#Yposition	Number of pixels that the top of the dialog box appears from the top of the screen. Specify -1 to have the dialog box appear in the center.
#width	Width of the window in pixels. Specify 0 to have the dialog box set its width automatically.
#height	Height of the window in pixels. Specify 0 to have the dialog box set its height automatically.
#modal	TRUE or FALSE. Sets whether the dialog box is modal.
#toolTips	TRUE or FALSE. Sets whether to use tooltips initially. (This is not supported in Version 1.0 of the MUI Xtra. It is reserved for future use.)
#closeBox	TRUE or FALSE. Specifies whether the dialog box has a close box.
#canZoom	TRUE or FALSE. Specifies whether the dialog box can zoom.

A Lingo statement can't contain more than 256 characters. Because lists are typically long, it might be necessary to assign some values to variables and then use those variables in the list.

Example These statements set up a list of overall dialog box properties.

- The first statement creates an instance of the MUI Xtra and assigns it to the variable theBox.
- The second statement assigns a list of predefined values to the variable aWindowPropList.
- The next three statements modify the name, callback, and width properties that were obtained from the GetWindowPropList function.
- The last statement displays the modified list in the Message window. The result appears at the end of the example.

```
set theBox = new(xtra "mui")
set aWindowPropList = GetWindowPropList(theBox)
set the name of    aWindowPropList = "General Settings"
set the callback of aWindowPropList =  "otherCallback"
set the width of    aWindowPropList = 200
put aWindowPropList
 -- [#type: #normal, #name: "General Settings", #callback: "otherCallback", #mode: #data,
#xPosition: 100, #yPosition: 120, #width: 200, #height: 210, #modal: 1, #toolTips: 0,
#closeBox: 1, #canZoom: 0]
```

Specifying dialog box content

To specify the content of a general purpose dialog box, create a linear list of definitions for each component of the dialog box. Each definition is a property list that defines one component. Components appear in the order that they are listed.

Some components define the structure of the dialog box, such as the beginning and end of the window and the start and end of horizontal and vertical sets of components. When constructing the dialog box, you must use this overall framework of components:

```
Window beginning
    Additional components as desired. This can include nested sets of additional horizontal
    and vertical groups, horizontal and vertical dividers, labels, and interface features such
    as buttons, check boxes, editable fields, and other interface elements.
Window end
```

To define individual components, create a property list that specifies each component. For convenience, use the GetItemPropList function obtain a predefined list of values and then modify properties as needed.

The following are properties for general dialog box components and their possible values:

Property	Possible value
#value	Determines the component’s value type. Possible types are integer, float, or string. See the following section "Possible #value settings for general purpose dialog box components" for more information about the #value property.
#type	One of the supported component types (see Possible component types)
#attributes	A list that specifies the component’s attributes. Attributes that you can specify depend on the component’s type. See Possible attribute settings
#title	String used as the title for the component. Specify ""for no title.
#tip	String used as the message in a tool tip. . Specify "" for no tool tip. (Tool tips aren’t supported in the initial release of the MUI Xtra. This is reserved for possible use in future releases.)
#locH	The distance of the component’s left edge from the left of the dialog box. There is no need to specify this property if #data is specified for the dialog box’s #mode property.
#locV	The distance of the component’s top from the top of the dialog box. There is no need to specify this property if #data is specified for the dialog box’s #mode property.
#width	The component’s width in pixels. Specify 0 to have Director automatically size the component’s width. There is no need to specify this property if #data is specified for the dialog box’s #mode property.
#height	The component’s height in pixels. Specify 0 to have Director automatically size the component’s height. There is no need to specify this property if #data is specified for the dialog box’s #mode property.
#enabled	TRUE or FALSE. Specify whether the item is enabled.

Possible #value settings for general purpose dialog box components

The #value settings for general purpose dialog box components are crucial for the correct display and editing of data in the dialog box. The following are possible settings for each type of component:

Component	Possible setting for #value	Examples
#dividerV	(Ignored)	(Not applicable)
#dividerH	(Ignored)	(Not applicable)
#bitmap	Cast member number, name, or reference	12, "First bitmap castmember", member 12 of castLib "Internal"
#checkBox	Boolean	TRUE, FALSE
#radioButton	Boolean	TRUE, FALSE
#PopupList	String, integer, or float	1, "Fred", 2.3
#editText	String	"Edit This Text"
#WindowBegin	(Ignored)	(Not applicable)
#WindowEnd	(Ignored)	(Not applicable)
#GroupHBegin	(Ignored)	(Not applicable)
#GroupHEnd	(Ignored)	(Not applicable)
#GroupVBegin	(Ignored)	(Not applicable)
#GroupVEnd	(Ignored)	(Not applicable)
#label	String	"label: ", "Long Label Text Here"
#IntegerSliderH	Integer	1, 100, 0
#FloatSliderH	Float	1.2, 0.0, 12.345
#defaultPushButton	(Ignored)	(Not applicable)
#cancelPushButton	(Ignored)	(Not applicable)
#pushButton	(Ignored)	(Not applicable)
#toggleButton	Boolean	TRUE, FALSE

Before opening the dialog box, use the Initialize command to specify which list Director uses as the source of definitions for the overall dialog box.

A Lingo statement can’t contain more than 256 characters. Because lists are typically long, it might be necessary to assign some values to variables and then use those variables in the list.

Example These statements specify the components for a general purpose dialog box that contains an editable field and a button. The dialog box is created from theBox, which is an instance of the MUI Xtra.

The first statement creates a new list named aWindowItemList.

Each subsequent set of statements obtains a predefined list of properties for an individual component, modifies it as necessary, and then adds the modified list to the overall list of dialog box components.

The final statement displays the new overall list in the Message window. The result appears at the end of the example. For easier reading, the display has been broken out into separate sections for each item in the list.

```
set aWindowItemList = []

-- Set up the beginning of the dialog box
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #windowBegin
append (aWindowItemList, duplicate(tempItemPropList))

-- Set up the beginning of an overall group
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #groupHBegin
append (aWindowItemList, duplicate(tempItemPropList))

-- Set up an editable field
-- This will use default values for text attributes, because no
-- list of values is assigned to the attributes property.
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #editText
set the value of tempItemPropList = "Enter text here"
append (aWindowItemList, duplicate(tempItemPropList))

-- Set up an OK button
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #pushButton
set the title of tempItemPropList = "OK"
append (aWindowItemList, duplicate(tempItemPropList))


-- Set up end of overall group
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #groupHBegin
append (aWindowItemList, duplicate(tempItemPropList))

-- Last, set up end of window
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #windowEnd
append (aWindowItemList, duplicate(tempItemPropList))

put aWindowItemList
-- [
[#value: 0, #type: #WindowBegin, #attributes: [], #title: "title", #tip: "tip", #locH: 20,
#locV: 24, #width: 200, #height: 210, #enabled: 1],
[#value: 0, #type: #GroupHBegin, #attributes: [], #title: "title", #tip: "tip", #locH: 20,
#locV: 24, #width: 200, #height: 210, #enabled: 1],
[#value: "Enter text here", #type: #editText, #attributes: [], #title: "title", #tip: "tip", #locH:
20, #locV: 24, #width: 200, #height: 210, #enabled: 1],
[#value: 0, #type: #pushButton, #attributes: [], #title: "OK", #tip: "tip", #locH: 20, #locV:
24, #width: 200, #height: 210, #enabled: 1],
[#value: 0, #type: #GroupHBegin, #attributes: [], #title: "title", #tip: "tip", #locH: 20,
#locV: 24, #width: 200, #height: 210, #enabled: 1],
[#value: 0, #type: #WindowEnd, #attributes: [], #title: "title", #tip: "tip", #locH: 20, #locV:
24, #width: 200, #height: 210, #enabled: 1]

]
```

Possible attribute settings

To specify the attributes of a general purpose dialog box component, assign a list of attribute specifications to the component's #attributes property.

What you can specify depends on the type of component. The following are possible properties and values you can specify for a component's attribute property. The available attributes vary depending on the type of component.

Attribute	Possible values	Can be set for
#textSize	One of the following: #large, #tiny, #normal(default)	Strings
#textStyle	A list that includes any of the following attributes: #bold, #italic, #underline, #plain (default), #inverse (v2)	Strings
#textAlign	One of the following: #left, #right, #center (defaults to system language standard)	Strings
#popupStyle	One of the following: #tiny, #cramped, #normal (default)	Pop-ups
#valueList	A list of values that appear in a pop-up. All values are coerced to strings. For example, Director treats ['one', #two, 3, 4.0] as four pop-up items, each of which is treated as a string.	Pop-ups
#valueRange	A list of a slider's minimum, maximum, increment, jump, and acceleration values. Available properties to set are #min, #max, #increment, #jump, and #acceleration. This list is the default setting: [#min:0.0, #max:1000.0, #increment:1.0, #jump:10.0, #acceleration:0.5]	Sliders
#sliderStyle	A linear list of one or both of #ticks or #value.	Sliders
#layoutStyle	A list of values that includes one or more of the following: #minimize, #lockPosition, #lockSize, #centerH, #right, #left, #centerV, #top, #bottom.	All items that aren't grouped by the #groupHBegin, #groupHEnd, #groupVBegin, or #groupVEnd properties.
#bitmapStyle	A property list that specifies that one of the bitmap icons in the MUI Xtra is to be used as an icon for a general purpose dialog box. The property in the property list is #bitmapIcon. Possible values are #stop, #note, #caution, #question, #error. For example, the statement set the attributes of tempPropItemList= [#bitmapIcon:#caution] sets the caution symbol as the value for #bitmapStyle.	Bitmaps

Example The following statement sets up attributes for an editable field:

```
set the attributes of tempTextItemList = [#textSize:#Normal, #textStyle:[#Normal]]
```

Example The following statement sets up attributes for a slider:

```
set the attributes of tempItemList = [#valueRange: [#min:0.0, #max:100.0, ¬
 #jump:5.0, #acceleration:0.5], #sliderStyle:[#ticks]]
```

Possible component types

The following are available component types, whether they use a title, and the attributes that you can specify for them. See Possible attribute settings [xref] for details about setting specific attributes.

Property	Title	Available attributes
#bitmap	No	#layoutStyle, #bitmapStyle
#cancelPushButton	Yes	#textSize, #layoutStyle
#checkBox	Yes	#textSize, #layoutStyle
#defaultPushButton	Yes	#textSize, #layoutStyle
#dividerH	No	#layoutStyle
#dividerV	No	#layoutStyle
#editText,	No	#textSize, #justification, #textStyle, #layoutStyle
#floatSliderH	No	#sliderStyle, #valueRange, #layoutStyle
#groupHBegin	No	None
#groupHEnd	No	None
#groupVBegin	No	None
#groupVEnd	No	None
#IntegerSliderH	No	#sliderStyle, #valueRange, #layoutStyle
#label	No	#textSize, #justification, #textStyle, #layoutStyle
#none	No	#layoutStyle
#popupList	No	#popupStyle, #valueList, #layoutStyle

Property	Title	Available attributes
#pushButton	Yes	#textSize, #layoutStyle
#radioButton	Yes	#textSize, #layoutStyle
#toggleButton	Yes	#textSize, #layoutStyle
#windowBegin	No	None
#windowEnd	No	None

The GetWidgetList function also returns a list of supported general purpose dialog box component types.

Sending dialog box events to Director

Sending dialog events back to Director lets you determine events such as clicking buttons, changing values, or when the dialog box moves or closes.

When a dialog box event occurs, the callback message includes three arguments:

- The callback event symbol, which identifies the type of event.
- Event-specific information such as the number in the #windowItemList of the item involved in the event
- The new item property list for the affected item

To specify how a general purpose dialog box responds, write Lingo that instructs the movie what to do in response to an event and then assign that Lingo to the event's symbol. Specify which handler responds to a dialog box event by assigning the handler's name to the dialog box's #callback property in the list assigned to the #windowPropList.

The following are possible events that a general purpose dialog box can send back to Director to indicate what happened to the dialog box:

Event	What occurred
#itemChanged	The item's value has changed.
#itemClicked	The user clicked an item in the dialog box.
#windowOpening	The dialog box window opened.
#windowClosed	The dialog box window closed.
#windowZoomed	The dialog box window was zoomed.
#windowResized	The dialog box window was resized.
#itemEnteringFocus	An item in the dialog box got focus.
#itemLosingFocus	An item in the dialog box lost focus.

These are some tricks for setting up a callback:

- ▶ Store item numbers in global variables when building an item list. To check whether an item is important, compare eventData to the stored index number.
- ▶ Because all buttons have their text in their titles, you can compare the title against text to determine if a particular button was clicked. For portability and easier localization, store text button titles in global string variables when setting up a dialog box and to do comparisons.
- ▶ Always make sure there is always a way to close a dialog box. Otherwise, it can be impossible to dismiss it.
- ▶ Because you can update items, if text changes you can enable an OK button that is unavailable until the text changes.
- ▶ Avoid hanging inside a modal dialog by making sure there is an event that can stop the dialog box.

Example The following handler contains a case statement that responds to an event in a general purpose dialog box. Each possible event is a condition that Director tests for.

The three arguments are the following:

- ▶ event is the type of event.
- ▶ eventData is specific information about the event.
- ▶ itemPropList is the list of property's for the item.

The handler's name is already assigned to the dialog box's #callback property in the list assigned to the #windowPropList.

```
on theWindowCallback event, eventData, itemPropList

  global gMuiSmileDialObj, smileyIndex
  if symbolP(event) then -- basic error check
    case event of
      #itemChanged :
        if the type of itemPropList = #IntegerSliderH then
          set smileyIndex = the value of itemPropList
          smileyUpdate  FALSE
        end if

      #itemClicked :
        case ( the type of itemPropList ) of
          #bitmap : beep
          #defaultPushButton :
            if ( the title of itemPropList = "Forward" ) then
              smileyIndexIncrement
              smileyUpdate TRUE
            end if
          #pushButton :
            if ( the title of itemPropList = "Back" ) then
              smileyIndexDecrement
              smileyUpdate TRUE
            end if

          #cancelPushButton :
            if ( objectP ( gMuiSmileDialObj ) ) then
              stop(gMuiSmileDialObj, 1)
            end if
        end case

        #windowOpening:
        #windowClosed:
        #windowZoomed :
        #windowResized :
        otherwise :
      end case
    end if
end theWindowCallback
```

Alert

Syntax Alert(*MUIObject*, *alertPropertiesList*)

This command displays an alert dialog box created from an instance of the MUI Xtra. This feature is in addition to the simple alerts generated by the alert command.

The MUI Xtra provides modal alerts. The alert can be moveable or non-moveable.

To create the alert, create a MUI Xtra object, and then issue the alert command with a list containing definitions of alert properties as the second parameter.

The following are properties that you must specify and their possible values:

Property	**Possible values**	**Specifies**
#buttons	#Ok #OkCancel #AbortRetryIgnore #YesNoCancel #YesNo #RetryCancel	The set of buttons that appear in the alert. The buttons appear in the order that they are named in each symbol.
#default	The ordinal number of the button that becomes the default. For example, if the alert's buttons are OK and Cancel, 2 specifies the Cancel button. Specify 0 for no default.	Which button is the default.
#icon	#stop #note #caution #question #error	The type of icon that appears in the alert. Specify 0 for no icon.
#message	A string	The message that appears in the alert
#movable	TRUE FALSE	Whether the alert is moveable
#title	A string	The alert's title

You must explicitly specify each of the alert's properties. The MUI Xtra doesn't provide a list of default alert properties.

Lingo returns a value for the button that the user clicks.

Alerts can be almost as big as the screen; you can display a lengthy description if appropriate.

Example The following statements create and display an alert dialog box.

- ▶ The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- ▶ The second statement sets up a list of the alert's properties.
- ▶ The final statements use the Alert command to display the alert and report which buttons the user clicks.

```
set alertObj = new(xtra "MUI")

set alertInitList = [ #buttons : #YesNo, ¬
#title : "Alert Test", #message : "This shows Yes and No buttons",¬
#movable : TRUE]
if objectP ( alertObj ) then
    set result = Alert( alertObj, alertInitList )
    case result of
        1 : -- the user clicked yes
        2 : -- the user clicked no
        otherwise : -- something is seriously wrong
    end case
end if
```

FileOpen

Syntax FileOpen(*MUIObject*, *string*)

This function displays a standard file open dialog box provided by an instance of the MUI Xtra.

The second parameter specifies a string that appears in the editable field when the dialog box opens. The user can specify which file to open by entering the file name in the editable field. When the user clicks a button, the text is returned.

- If the user clicks Cancel, the returned text is the same as the value that was passed in.
- If the user clicks OK, the returned text is a platform-specific path.

Example These statements create and display a standard file open dialog box.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement assigns a string to the variable fileString, which is used later as the second parameter of the FileOpen command.
- The third statement uses the FileOpen command to generate the open file dialog box.
- The final statements check whether the original string sent with the FileOpen command is the same as the string that was returned when the user clicked a button. If the values are different, the user selected a file to open.

```
set aMuiObj = new (xtra "MUI")

set fileString = "Open this file"

set result = fileOpen(aMuiObj, fileString)

-- Check to see if the dialog was canceled
if (result < > fileString) then
    -- The dialog wasn't canceled, do something with
-- the new path data.
else
    put "ERROR - fileOpen requires a valid aMuiObj"
end if
```

FileSave

Syntax FileSave(*MUIObject*, *string*, *message*)

This function displays a standard file saving dialog box that saves the current file. The dialog box is created from an instance of the MUI Xtra.

- The *string* parameter specifies the string that appears in the dialog box's file name field. The user can use this field to enter a new file name for the file. When the user clicks a button, Lingo returns a value for string that contains the field's content. If the user clicks Cancel, the returned string is the same as the original string.
- The *message* parameter is the string that appears above the dialog box's editable field.

Example These statements create and display a file save dialog box.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement assigns a string to the variable *fileString*, which is used later as the second parameter of the FileSave command.
- The third statement uses the FileSave command to generate the save file dialog box.
- The final statements check whether the result after the user clicks a button is the same as the string sent when the dialog box opened. If it differs, the user clicked something other than Cancel.

```
set aMuiObj = new (Xtra "MUI")

set fileString = "save this file"

set result = fileSave( aMuiObj, fileString, "with this prompt" )

if (result < > fileString) then
    -- The dialog wasn't canceled, do something with the new
    - path data.
end if
```

GetItemPropList

Syntax GetItemPropList(*MUIObject*)

This function returns a list of the MUI Xtra's predefined properties for components in a general purpose dialog box. It is useful for defining new components in a general purpose dialog box. Use GetItemPropList to obtain a comprehensive list of properties and values and then edit individual properties as necessary.

The list of properties and values are the following:

Property	Default value
#value	0
#type	#checkBox
#attributes	[],
#title	"title"
#tip	"tip" (This is reserved for possible use in later versions of the MUI Xtra.)
#locH	20
#locV	24
#width	200
#height	210
#enabled	1

Example These statements define the beginning of a dialog box window.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement assigns a list of default dialog component settings to the variable tempItemProps.
- The third statement makes the component the dialog box's beginning by changing its type to #windowBegin.

```
set aMuiObj = new (Xtra "MUI")
set tempItemProps = GetItemPropList(aMuiObj)

set the type of tempItemProps = #windowBegin
```

GetUrl

Syntax GetUrl(*MUIObject*, *message*, *MovableOrNot*)

This function displays a dialog box for entering a URL and returns the URL that the user enters.

- ▸ *message* specifies the message that appears in the field for entering a URL. When the dialog box is first opened, this string is sent as a predefined value. When the user clicks a button, Lingo returns the string that the user entered. If the user clicks Cancel, the returned string is the same as the original value.
- ▸ On the Macintosh, *MovableOrNot* specifies whether the dialog box is movable. TRUE makes the dialog box movable. FALSE makes the dialog box not movable. The GetURL dialog box is always movable in Windows.

Example These statements display a dialog box for entering a URL.

- ▸ The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- ▸ The second statement uses the GetUrl function to display a moveable dialog box for entering URLs and assigns the dialog box to the variable result. The message "Enter a URL here" appears in the dialog box's field for entering a URL.
- ▸ The final statements check whether the result after the user clicks a button is the same as the string sent when the dialog box opened. If it differs, the user entered a URL and clicked OK.

```
set MUIObj = new (xtra "Mui")

set result = GetUrl(MUIObj, "Enter a URL", TRUE )

if objectP ( MUIObj) then
    set result = GetUrl( MUIObj, "Enter a URL", TRUE )
    if ( result < > "Enter a URL" ) then
        goToNetPage result
    end if
end if
```

GetWidgetList

Syntax GetWidgetList(*MUIObject*)

This function returns a linear list of symbols for types of general purpose dialog box components supported for an instance of the MUI Xtra.

Example This statement displays a list of widgets supported by MUIObject, which is an instance of the MUI Xtra:

```
put GetWidgetList(MUIObject)
```

```
-- [#dividerV, #dividerH, #bitmap, #checkBox, #radioButton, #PopupList, #editText,
#WindowBegin, #WindowEnd, #GroupHBegin, #GroupHEnd, #GroupVBegin,
#GroupVEnd, #label, #IntegerSliderH, #FloatSliderH, #defaultPushButton,
#cancelPushButton, #pushButton, #toggleButton]
```

GetWindowPropList

Syntax GetWindowPropList(*MUIObject*)

This function returns a list of the MUI Xtra’s predefined settings for a general purpose dialog box’s window.

When defining a new general purpose dialog box, use GetWindowPropList function to obtain a comprehensive list of dialog box properties and values and then edit individual properties as necessary. Besides being more convenient, this technique ensures compatibility with future versions of the MUI Xtra that may have additional properties.

These are the window properties and predefined values that GetWindowPropList returns:

Property	Predefined value
#type	#normal
#name	"window"
#callback	"nothing"
#mode	#data
#xPosition	100
#yPosition	120
#width	200
#height	210
#modal	1
#toolTips	0

Property	Predefined value
#closeBox	1
#canZoom	0

Example These statements define a new general purpose dialog box. The first statement assigns a list of predefined properties to the variable thePropList. Subsequent statements customize the dialog box by modifying these settings:

```
set thePropList = GetWindowPropList(muiObject)

set the name     of thePropList = "Picture Window"
set the callback of thePropList = "theWindowCallback"
set the mode     of thePropList = #data
set the modal    of thePropList = TRUE
set the closeBox of thePropList = FALSE
```

Initialize

Syntax Initialize (*MUIObject*, *initialPropertyList*)

This command sets up a general purpose dialog box from an instance of the MUI Xtra. *initialPropertyList* is a property list that specifies where Director obtains definitions for the dialog box's attributes.

- The property list associated with the #windowPropList property (see Specifying overall dialog box properties) is the list Director uses for definitions of the overall dialog box's attributes.
- The linear list associated with the #windowItemList property (see Specifying dialog box content) is the list Director uses for definitions of individual components. Each item in the list is a property list that defines one component.

Example This statement initializes a general purpose dialog box created from MUIObject, which is an instance of the MUI Xtra. The list aWindowPropList contains definitions for the overall dialog box. The list aWindowItemList contains definitions for the dialog box's individual components:

```
Initialize(MUIObj, [#windowPropList:aWindowPropList, ¬
 #windowItemList:aWindowItemList])
```

ItemUpdate

Syntax ItemUpdate(*MUIObject*, *itemNumber*, *itemInputPropList*)

This command updates a component in a general purpose dialog box. It is useful for updating a dialog box in response to user actions while the dialog box is displayed.

- *itemNumber* represents the number of the item being updated.
- *itemInputPropList* represents the list of new properties for the item.

The ItemUpdate command can be used for many things; possible uses include enabling or disabling buttons, changing the range of a pop-up, updating a sliders position, and updating editable text items if the user enters an invalid value.

You may want to update individual items in a dialog box depending on user input, user interaction, or to display underlying data. Although you would typically update an item's #value, you can also update everything else about an item, except for its type. Set the height, width, locH, and locV properties to -1 to keep their current values.

Example These statements update the dialog box component that has the number itemNum.

- The first statement obtains the component's definitions from the overall list of item definitions.
- The second and third statements modify the component's type and attribute properties.
- The last statement uses the ItemUpdate command to update the component's settings.

```
set baseItemList = getAt ( theItemList, itemNum )
set the type of baseItemList  = #IntegerSliderH
set the attributes  of baseItemList = [#valueRange  :[#min:1, #max:8, #increment:1,
#jump:1, #acceleration:1]
ItemUpdate(MUIObj, itemNum, baseItemList)
```

Example This handler updates [???]

```
on smileyUpdate
    -- declare globals
    global smileyIndex, gMuiSmileDialObj, itemNumSmile, ¬ itemNumSlide, smileItemList

    -- validate dialog object
    if ( objectP ( gMuiSmileDialObj ) ) then
        -- get a list to put in new/updated values
        set baseItemList = duplicate ( getAt ( smileItemList, ¬
        itemNumSmile ) )

        -- metrics can be set to -1, this "keeps them the same"
        -- instead of updating.
        -- could also be set to a new value if you
        -- wanted to resize the item or relocate it.
        set the width of baseItemList  = -1 -- keep previous
        set the height of baseItemList = -1 -- keep previous
        set the locH of baseItemList   = -1 -- keep previous
        set the locV of baseItemList   = -1 -- keep previous

        -- in this particular case, the value is
        -- the only thing that's changing
        set the value of baseItemList  = string(smileyIndex)
        -- member name

        -- tell the dialog to update the item number
        -- with the new item list
        ItemUpdate(gMuiSmileDialObj, itemNumSmile, baseItemList)
    end if
end
```

new

Syntax new(xtra "MUI")

```
set theObject = new(xtra "MUI")
```

This function creates a new instance of the MUI Xtra that you can use as a dialog box. This use is in addition to creating child objects, new cast members, and instances of other Xtras with the new function.

You must create a new instance of the MUI Xtra before you can use it to create a dialog box or obtain any of the Xtra's predefined values.

Example This statement creates a new instance of the MUI Xtra and assigns it to the variable MUIObject:

```
set MUIObject = new(xtra "MUI")
```

run

Syntax run(*MUIObject*)

This command displays a general purpose modal dialog box created from an instance of the MUI Xtra.

Before Director can open the dialog box, use the Initialize command to define the dialog box. See Specifying dialog box content and Specifying Overall dialog box properties for more information about specifying the content of a general purpose dialog box.

To open a non-modal dialog box, use the WindowOperation command with the #show option. This command allows other Lingo to run in the movie while the non-modal dialog box is open.

Example This handler checks whether the object MUIObject exists and displays a general purpose dialog box from MUIObject if it is:

```
on runDialog
    global MUIObject
    if objectP(MUIObject) then
            run(MUIObject)
    end if
end
```

stop

Syntax stop(*MUIObject*, *stopItem*)

This function stops the general purpose dialog created from an instance of the MUI Xtra. After the function is called, Lingo returns the results as the number for the *stopItem* parameter.

The *stopItem* parameter is returned from the run(MUIOject) call. Use this to pass back a parameter indicating how the dialog box was stopped. For example, this could return 1 if the user clicked OK and return 0 if the user clicked Cancel.

Note: To close a non-modal dialog box, use the WindowOperation command with the #hide option.

Example This handler stops the general purpose dialog box created from MUIObject. The second parameter of the stop command is zero, which fulfills the requirement for a value but has no other purpose:

```
on stopDialog
        global MUIObject
        if ( objectP (MUIObject)) then
                stop(MUIObject, 0)
        end if
end stopDialog
```

WindowOperation

Syntax WindowOperation(*MUIObject*, *operation*)

This command controls the window for a general purpose dialog box.

Replace the operation parameter with a value that determines what the window does. Possible values and their result are:

Possible values	Result
#show	Displays a non-modal dialog box only. (To open a modal dialog box, use the Run command.)
#hide	Hides a non-modal dialog box. (To close a modal dialog box, use the Stop command.)
#center	Centers the window on the monitor screen.
#zoom	Sends a message that the user clicked the zoom box on the window. The callback handler must resize the dialog box, if you want the window to resize after the user clicks the zoom box.
#tipsOn	Turns tool tips on. (This is reserved for future versions of the MUI Xtra.)
#tipsOff	Turns tool tips off. (This is reserved for future versions of the MUI Xtra.)

Example This handler checks whether MUIObject exists and displays the dialog box if the does:

```
on showDialog
        global MUIObject
        if objectP( MUIObject ) then
                WindowOperation( MUIObject, #show )
        end if
end showDialog
```

Example This statement hides the dialog box created from MUIObject:

```
WindowOperation(MUIObject, #hide)
```