Preamble

The Objective-X (name subject to change) class library is designed to bring the ease of GUI programming and machine independence that is familiar to programmers of the NeXT computer to the X Window System. It is written in Objective-C for maximum flexibility. This document describes the preliminary classes and method calls as precisely as possible to assist in the development in this library.

There are two motivations for this library:
One, the X Window System is so complex, machine dependent, and sometimes restricting, that it can be difficult to write quality graphical interfaces. This library is, as much as physically possible, machine independent. The same code to produce the graphics on an 8-bit, 4Mb memory machine without backing store will work on a 24-bit, 128Mb machine with backing store and other features, while taking advantage of the greater power of the larger machine.

Two, I believe in a graphical interface that is customizable to the user to the extent that the same software package with two different users may not even resemble each other. This should be possible without having to write the code to support the different users' preferences. This is extremely difficult to perform using only defaults files. The way I propose to do it is through an interface builder and remote object calling. The interface construction file (**.icf** file, equivalent to a **.nib** file in NeXTStep) is separate from the main program body, and will run in a separate process, relieving the application from having to deal with the events in the interface at a low level. When an event that the application shouldn't have to bother with occurs, (example, the "Info..." button in the main menu: it just pops up a panel, which can be done without the application's intervention) the application isn't notified, the state of the interface is simply changed to handle the info panel. This allows for a "multi-threading" of sorts, causing the application to be interrupted only when necessary.

The main application code (core app) would consist of a kernel that would deal with the internal data structures and organization of it's output information, and maybe even an internal event loop (a telecom application that was processing a download would have to send ACKs on occasion); and a set of remote object hooks (generally a target/action device or a delegate). When the output changes, the Interface Handler would be notified, and the display updated. When a user generates an event the application needs to know about, one of it's hooks (as defined by the **.icf**) would be called through the remote object system. The processing of this event on the application side would be transparent to the application in that it can occur though the generation of a signal, which is handled quickly to allow the application to get back to it's regular work.

The interface itself is distributed in an **.icf** file. This file would contain the interface built by the application's author, showing a working set of object-target/action-delegate connections. The user is free to load the **.icf** into the interface builder and change it in whatever way suits him. A spreadsheet core and a word processor core could be connected into a single interface to allow the user access to both simultaneously, and it would look and feel like they were both the same app. A live video recording program and a video editing package could be seamlessly integrated.

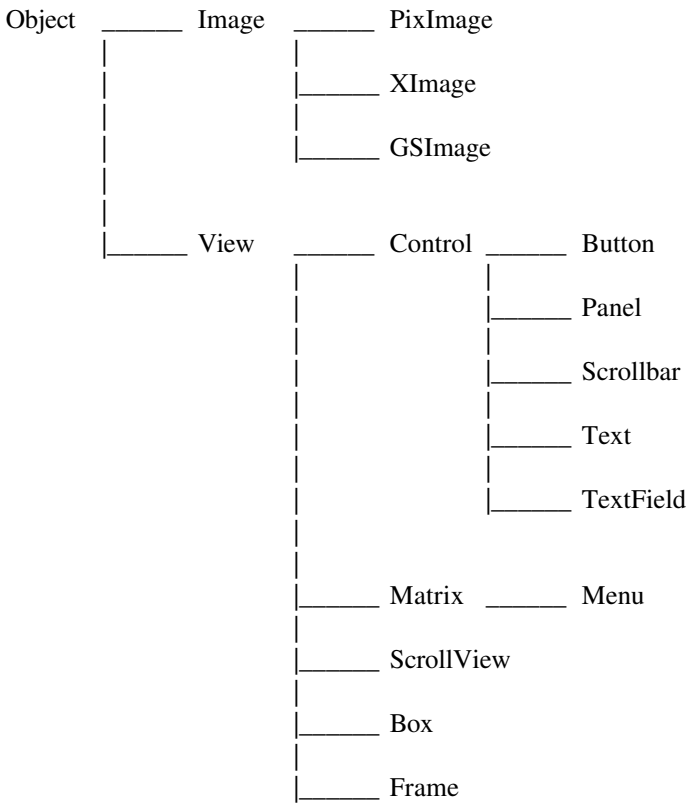
This form of software development could even allow commercial companies to write regular application code and interfaces separately so that the user still has a great deal of control over their apps without any effort on the part of the developer. Only the hooks' method names would be known to the interface, and that's all that would be needed.

This system should also be flexible enough for a set of utility libraries to be available to a non-programmer to essentially write their own programs without learning a programming language. For example, if a math library, spreadsheet, or even a 3-D CAD package were already available, the non-programmer could make the appropriate connections to the hooks in those libraries to make a fully functional scientific calculator.

The class hierarchy is on the next page.

Class Hierarchy

This is currently the entire class hierarchy. Changes are still planned. Comments are requested. My apologies if it doesn't appear properly formatted... I did this in a hurry.



Information

Note: the class descriptions that follow are still incomplete. I'd like to hear some feedback as to the design of the system, but please realize that some obvious things, such as instance variables needed to store important information or methods needed to supplement other methods, may have been left out. Also, in some cases the class description is very thorough, and in some it is only a couple of words. This is due to the fact that I'm only able to work on this on the weekend, and I'm rushing to get this document out to the public. I am rushing even more because I intend to start actual programming by 23-Oct-93.

Thanks for understanding.

Planned enhancements

I would like to add particular panels, such as Print, File (open/save), Color, and Font. I would also like to have printer drivers, fax cabability, and possibly some changes to the Image hierarchy to allow for querying fonts and drawing cabailities so that new Image subclasses can be used by existing View classes for text and icons without using categories.

In the near future, I'd need to add the remote object system.

In the more distant future, I'd like to add support for 3-D drawing and Sound.

As some preliminary applications, there is of course the Interface Builder, a word processor on the level of Microsoft Word or WordPerfect, a spreadsheet on the level of Microsoft Excel or Lotus Improv if we can get by Lotus' threat to sue anyone who uses the Improv layout style (doubtful!).

Class: Box

Inherits From: View
Declared In: objx/Box.h

Class Description

Draws a 3-D or flat box-looking frame around a set of controls. May contain a title.

Instance Variables

char *	**text**
Font	**font**
enum alignment	**align**
enum position	**location**
enum framestyle	**bordstyle**
int	**bordthick**

text	The string to be displayed on the frame of the box.
font	The X font of the above text.
align	The alignment of the text (left, right, center)
location	The location of the text (top, bottom, left, right)
bordstyle	The style of the frame used as the box (flat, 3-D in, 3-D out). This may just be contained in the frame object.
bordthick	The number of pixels thick to draw the box.

Method Types

Displaying text	- getXFont - getXText - setXFont: - setXText:
Text placement	- getTextAlignment - getTextLocation - setTextAlignment: - setTextLocation:
Border appearance	- getBorderThickness - getBorderStyle - getBorderInColor - getBorderInImage - getBorderOutColor - getBorderOutImage - setBorderThickness: - setBorderStyle: - setBorderInColor: - setBorderInImage: - setBorderOutColor: - setBorderOutImage:

Class Methods

None

Instance Methods

- (Color)**getBorderInColor**

Returns the color of the inward facing border.

- getBorderInImage

Returns the Image being used as the background pixmap for the inward border.

- (Color)**getBorderOutColor**

Returns the color of the outward facing border.

- getBorderOutImage

Returns the Image being used as the background pixmap for the outward border.

- (FrameStyle)**getBorderStyle**

Returns the style of the frame. See **setBorderStyle** for frame types.

- (short)**getBorderThickness**

Returns the thickness of each border, in pixels.

- (struct Alignments *)**getTextAlignment**

Returns the alignments being used for the text in relation to the entire Box object. (Does this make the "textLocation" calls unnecessary?) See the Information page about alignments.

- (Position)**getTextLocation**

Returns the current position of the text. See **setTextLocation** for more information.

- getXFont

Returns the Font object being used to draw the text, or nil if there is none.

- (char *)**getXText**

Returns the text being used in Box, or NULL if there is none.

- setBorderInColor:(Color)*clr*

Sets the color of the inward (dark shadowed) border. Returns *self*.

- setBorderInImage:*image*

Sets the Image of the inward (dark shadowed) border. The pixmap contained within the Image is set to be the background pixmap in the shaped window for the appropriate borders. Returns *self*.

- setBorderOutColor:(Color)*clr*

Sets the color of the outward (light shadowed) border Returns *self*.

- setBorderOutImage:*image*

Sets the Image of the outward (light shadowed) border. The pixmap contained within the Image is set to be the background pixmap in the shaped window for the appropriate borders. Returns *self*.

- setBorderStyle:(FrameStyle)*type*

Sets the type of border to be displayed. Current options are: `FRAME_IN`, `FRAME_OUT`, `FRAME_INOUT` and `FRAME_OUTIN`. Returns *self*.

- setBorderThickness:(short)*thick*

Sets the thickness of each side of the border. Returns *self*.

- setTextAlignment:(struct Alignments *)*align*

Sets the alignment of the text. See the Information page regarding alignments. Returns *self*.

- setTextLocation:(Position)pos

Sets the location of the text for the box to be one of `TEXT_TOP`, `TEXT_ABOVETOP`, `TEXT_BELOWTOP`, `TEXT_BOTTOM`, `TEXT_ABOVEBOTTOM`, or `TEXT_BELOWBOTTOM` Returns *self*.

- setXFont:*font*

Sets the font to be used by the text XImage to draw the Box's text. Returns *self*.

- setXText:(char *)*string*

Sets the string to be used by the text XImage to draw the Box's text. Returns *self*.

Class: Button

Inherits From: Control
Declared In: objx/Button.h

Class Description

Handles standard push and toggle button events and callbacks. May contain a title, icon, or other subimages subject to a few restrictions.

Instance Variables

char *	**text**
char *	**text2**
Font	**font**
Font	**font2**
enum btntype	**type**

text	The string to be displayed when the button is "unset" (value 0).
text2	The string to be displayed when the button is "set" (value 1).
font	Font to use for the first string.
font2	Font to use for the second string.
type	The way the button highlights or shows it's alternate graphics (momentary/toggle, show alt graphics, show light, others.?.?.)

Method Types

Displaying text	- getPrimaryXFont - getPrimaryXText - getSecondaryXFont - getSecondaryXText - getPrimaryIcon - getSecondaryIcon - setPrimaryXFont: - setPrimaryXText: - setSecondaryXFont: - setSecondaryXText: - setPrimaryIcon: - setSecondaryIcon:
Text placement	- getPrimaryTextAlignment - getSecondaryTextAlignment - getPrimaryIconAlignment - getSecondaryIconAlignment - setPrimaryTextAlignment: - setSecondaryTextAlignment: - setPrimaryIconAlignment: - setSecondaryIconAlignment:

Button type	- getType - setType:

Class Methods

None

Instance Methods

- (struct Alignments *)**getPrimaryIconAlignment**

Returns the alignments for the icon when the Button's value is 0.

- (struct Alignments *)**getPrimaryTextAlignment**

Returns the alignments for the text when the Button's value is 0.

- getPrimaryXFont

Returns the font used for the text when the Button's value is 0.

- getPrimaryXIcon

Returns the Image used for the Button when it's value is 0.

- (char *)**getPrimaryXText**

Returns the string used for the text when the Button's value is 0.

- (struct Alignments *)**getSecondaryIconAlignment**

Returns the alignments for the icon when the Button's value is 1.

- (struct Alignments *)**getSecondaryTextAlignment**

Returns the alignments for the text when the Button's value is 1.

- getSecondaryXFont

Returns the font used for the text when the Button's value is 1.

- getSecondaryXIcon

Returns the Image used for the Button when it's value is 1.

- (char *)**getSecondaryXText**

Returns the string used for the text when the Button's value is 1.

- (ButtonType)**getType**

Returns the type of button being used. Button types are still to be determined.

- setPrimaryIconAlignment:(struct Alignments *)*align*

Returns the alignments for the icon when the Button's value is 0.

- setPrimaryTextAlignment:(struct Alignments *)*align*

Returns the alignments for the text when the Button's value is 0.

- setPrimaryXFont:*font*

Returns the font used for the text when the Button's value is 0.

- setPrimaryXIcon:*image*

Returns the Image used for the Button when it's value is 0.

- setPrimaryXText:(char *)string

Returns the string used for the text when the Button's value is 0.

- setSecondaryIconAlignment:(struct Alignments *)*align*

Returns the alignments for the icon when the Button's value is 1.

- setSecondaryTextAlignment:(struct Alignments *)*align*

Returns the alignments for the text when the Button's value is 1.

- setSecondaryXFont:*font*

Returns the font used for the text when the Button's value is 1.

- setSecondaryXIcon:*image*

Returns the Image used for the Button when it's value is 1.

- setSecondaryXText(char *)string

Returns the string used for the text when the Button's value is 1.

- setType:(ButtonType)*type*

Returns the type of button being used. Button types are still to be determined.

Class: Control

Inherits From: View
Declared In: objx/Control.h

Class Description

This is the super class of all objects that use events. It contains the entire target/action and delegate calling mechanism. It is not designed to be used by itself; it should be subclassed.

Instance Variables

id	**fwdtab**
id	**revtab**
id	**rightarr**
id	**leftarr**
id	**uparr**
id	**downarr**
double	**dvalue**
long	**lvalue**
char *	**svalue**
TargAct *	**talist**
Delegate	**dellist**
BOOL	**targenabled**
BOOL	**delegenabled**

fwdtab, revtab	The object the keyboard focus is changed to if tab or shift tab is pressed.
rightarr,leftarr, uparr, downarr	The object the keyboard focus is changed to if an arrow key is pressed.
dvalue, lvalue, svalue	Control's value. Should probably just be a double.
talist	List of active target/action/type sets.
dellist	List of active delegates.
targenabled	Whether or not targets will be called.
delenabled	Whether or not delegates will be called.

Method Types

Responding to events	- usesEvents - setUseEvents: - getChildAtPoint: - acceptEvent: - acceptMouseEvent: - acceptKeyEvent: - callTargetsWithActionType: - callDelegatesWithSelector:
Image location	- setBoundRect:notify: - setClipRect:notify:

Changing keyboard focus	- getFwdTab - getRevTab - getRightArrow - getLeftArrow - getUpArrow - getDownArrow - setFwdTab:mirror: - setRevTab:mirror: - setRightArrow:mirror: - setLeftArrow:mirror: - setUpArrow:mirror: - setDownArrow:mirror:
Control's value	- getLongValue - getUnsignedLongValue - getDoubleValue - getStringValue - setLongValue: - setUnsignedLongValue: - setDoubleValue: - setStringValue: - takeLongValueFrom: - takeUnsignedLongValueFrom: - takeDoubleValueFrom: - takeStringValueFrom:
Mouse tracking	- getMouseTrackingRectWhenUp - getMouseTrackingRectWhenDown - setMouseTrackingWhenUpInRect: - setMouseTrackingWhenDownInRect:
Border appearance	- isBordered - setBordered: - isBorderBezeled - setBorderBezeled: - getBorderThickness: - setBorderThickness: - getBorderInColor - getBorderInImage - getBorderOutColor - getBorderOutImage - setBorderInColor: - setBorderInImage: - setBorderOutColor: - setBorderOutImage: - exchangeBorders
Target/Action and delegates	- getTargetActionList - addTarget:action:type: - removeTarget:action:type: - areTargetsEnabled - setTargetsEnabled: - getDelegateList - addDelegate: - removeDelegate:

- areDelegatesEnabled
- setDelegatesEnabled:

- *doubleValueWillChange:from:to:wasTaken:*
- *stringValueWillChange:from:to:wasTaken:*
- *doubleValueChanged:from:to:wasTaken:*
- *stringValueChanged:from:to:wasTaken:*
- *keyPressed:key:mouseat:*
- *keyReleased:key:mouseat:*
- *mouseMoved:from:to:buttons:*
- *mousePressed:buttons:at:*
- *mouseReleased:buttons:at:*
- *mouseClicked:buttons:at:num:*
- *mouseEntered:at:*
- *mouseExited:at:*
- *boundRectWillChange:from:to:*
- *viewRectWillChange:from:to:*
- *boundRectChanged:from:to:*
- *viewRectChanged:from:to:*

Class Methods

None

Instance Methods

- acceptEvent:(struct Event *)*event*

Called by the object's parent. Events are either handled or passed along to **acceptMouseEvent** or **acceptKeyEvent**. Returns *self*.

- acceptKeyEvent:(struct Event *)*event*

Called by **acceptEvent**. Subclasses use this method to determine what action to take. Returns *self*.

- acceptMouseEvent:(struct Event *)*event*

Called by **acceptEvent**. Subclasses use this method to determine what action to take. Returns *self*.

- addDelegate:*anObject*

Adds *anObject* to the list of delegates. Returns *self*.

- addTarget:*anObject*
action:(SEL)*aSelector*
type:(eventType)*type*

Adds a target/action item to be called when the control received an event of type *type*. *anObject* must respond to *aSelector* and *aSelector* must take one argument, that being *sender* (which would be *self* from the control's point of view). MultiClick is assumed to be NO (see **addTarget:action:multiClick**).

- (BOOL)**areDelegatesEnabled**

Returns whether or not delegates are being called.

- (BOOL)**areTargetsEnabled**

Returns whether or not targets are being called.

- (BOOL)**callTargetsWithActionType:**(eventType)*type*

Looks through the list of target/action items and calls all of those whose eventType match *type*.

- (BOOL)**callDelegatesWithSelector:**(SEL)*sel*

Skims the list of delegates and calls all of those that respond to *sel*. *sel* must be a known selector to the delegates system. See the delegate methods at the end of this class description.

- exchangeBorders

Switches the colors or Images of the upper left and lower right borders. Change is only visible if isBezeled is true. Returns *self*.

- (Color)**getBorderInColor**

Returns the color used for the lower right edge of the bezel or NULL if an Image is being used.

- getBorderInImage

Returns the Image used for the lower right edge of the bezel, or nil if one is not defined.

- (Color)**getBorderOutColor**

Returns the color used for the lower right edge of the bezel or NULL if an Image is being used.

- getBorderOutImage

Returns the Image used for the lower right edge of the bezel, or nil if one is not defined.

- (short)**getBorderThickness**

Returns the border's thickness in pixels.

- getChildAtPoint:(const point *)*where*

Returns the id of the object that exists connected to the Control at the point *where*, or nil if there isn't one.

- (rect *)**getClipRect**

Returns the current visible rectangle.

- (delegateList *)**getDelegateList**

Returns the list of delegates.

- getDoubleValue

Returns the value of the Control as a double.

- getDownArrow

Returns the object that will have the keyboard focus after the user presses the down arrow.

- (Color)**getForegroundColor**

Returns the current color of the text and other single color Images in the Control.

- getFwdTab

Returns the object that will have the keyboard focus after the user presses the tab key.

- getLeftArrow

Returns the object that will have the keyboard focus after the user presses the left arrow.

- (long)**getLongValue**

Returns the value of the Control as a long. If the value is out of bounds of a long, the value returned will be cast in the normal manner. If the value contains a fractional portion, it will be rounded.

- (rect *)**getMouseTrackingRectWhenDown**

Returns the rectangle from which **mouseMoved:from:to:buttons:** delegate calls will occur when a mouse button is down and inside it.

- (rect *)**getMouseTrackingRectWhenUp**

Returns the rectangle from which **mouseMoved:from:to:buttons:** delegate calls will occur when a mouse button is up and inside it.

- getRevTab

Returns the object that will have the keyboard focus after the user presses shift-tab.

- getRightArrow

Returns the object that will have the keyboard focus after the user presses the right arrow.

- (char *)**getStringValue**

Returns the value of the Control as a string.

- (targActList *)**getTargetActionList**

Returns the list of target/action pairs in use for this Control.

- (unsigned long)**getUnsignedLongValue**

Returns the value of the Control as an unsigned long. If the value is out of bounds of an unsigned long, the value returned will be cast in the normal manner. If the value contains a fractional portion, it will be rounded.

- getUpArrow

Returns the object that will have the keyboard focus after the user presses the up arrow.

- (BOOL)**isBorderBezeled**

Returns whether the border is a bezel or flat. If the border is flat, the border's color/Image will be the one set for the upper left edge, and the lower right color/Image is ignored..

- (BOOL)**isBordered**

Returns whether or not there is a border around the Control.

- removeDelegate:*anObject*

Removes *anObject* from the list of delegates. Returns *self*.

- removeTarget:*anObject*
action:(SEL)*aSelector*
type:(eventType)*type*

Removes the target and action pair from the target/action list. Returns *self*.

- setBorderBezeled:(BOOL)*bezeled*

Sets the border of the Control to be a bezeled one. This value is not used if **isBordered** is NO. Returns *self*.

- setBordered:(BOOL)*bordered*

Sets whether or nor a border is seen. The appearance of the border is determined by **isBezeled**. Returns *self*.

- setBorderInColor:(Color)*color*

Sets the color used in the upper right edge of the Control's border. Replaces the border Image, if any. Returns *self*.

- setBorderInImage:*image*

Sets the Image used in the upper right edge of the Control's border. Replaces the border color, if any. Returns *self*.

- setBorderOutColor:(Color)*color*

Sets the color used in the lower right edge of the Control's border. Replaces the border Image, if any. Ignored if there is no border or the border is not bezeled. Returns *self*.

- setBorderOutImage:*image*

Sets the Image used in the lower right edge of the Control's border. Replaces the border color, if any. Ignored if there is no border or the border is not bezeled. Returns *self*.

- setBorderThickness:(short)*pixels*

Sets the thickness of the border in pixels. Returns *self*.

- setBoundRect:(const rect *)*rect*
notify:(BOOL)*notify*

Sets the physical size (and location) of the Control. Normally an application will use attachments to determine size and location. *notify* determines whether or not the delegates are informed. Returns *self*.

- setClipRect:(const rect *)*rect*

Clips the control's rectangle to *rect* and requests that its parent recompute its location within the parent's view. Returns *self*.

- setDelegatesEnabled:(BOOL)*enable*

Sets whether or not delegates are called upon occurance of keyboard and mouse events. Returns *self*.

- setDoubleValue:(double)*value*

Sets the value of the Control as a double. Returns *self*.

- setDownArrow:*aControl* **mirror:**(BOOL)*doMirror*

Sets the object that will have the keyboard focus after the user presses the down arrow. *aControl* must have the same Panel ancestor as *self*. If *doMirror* is YES, *aControl* will be set to have the up arrow return the focus here, unless the Control has another use for the up arrow (Text is a good example). Returns *self*.

- setFwdTab:*aControl* **mirror:**(BOOL)*doMirror*

Sets the object that will have the keyboard focus after the user presses the tab key. *aControl* must have the same Panel ancestor as *self*. If *doMirror* is YES, *aControl* will be set to have shift-tab return

the focus here, unless the Control has another use for the shift-tab sequence (Text is a good example). Returns *self*.

- setLeftArrow:*aControl* **mirror:**(BOOL)*doMirror*

Sets the object that will have the keyboard focus after the user presses the left arrow. *aControl* must have the same Panel ancestor as *self*. If *doMirror* is YES, *aControl* will be set to have the right arrow return the focus here, unless the Control has another use for the right arrow (Text is a good example). Returns *self*.

- setLongValue:(long)*value*

Sets the value of the Control as a long. Returns *self*.

- setMouseTrackingWhenDownInRect:(const rect *)*rect*

Sets whether or not delegates are notified of mouse movements when any mouse button is down in the specified *rect*. Returns *self*.

- setMouseTrackingWhenUpInRect:(const rect *)*rect*

Sets whether or not delegates are notified of mouse movements when any mouse button is up in the specified *rect*. Returns *self*.

- setRevTab:*aControl* **mirror:**(BOOL)*doMirror*

Sets the object that will have the keyboard focus after the user presses the shift-tab keys. *aControl* must have the same Panel ancestor as *self*. If *doMirror* is YES, *aControl* will be set to have tab return the focus here, unless the Control has another use for the tab key (Text is a good example). Returns *self*.

- setRightArrow:*aControl* **mirror:**(BOOL)*doMirror*

Sets the object that will have the keyboard focus after the user presses the right arrow. *aControl* must have the same Panel ancestor as *self*. If *doMirror* is YES, *aControl* will be set to have the left arrow return the focus here, unless the Control has another use for the left arrow (Text is a good example). Returns *self*.

- setStringValue:(const char *)*value*

Sets the value of the Control as a string. Returns *self*.

- setTargetsEnabled:(BOOL)*enable*

Sets whether or not targets are notified of mouse and keyboard events. Returns *self*.

- setUnsignedLongValue:(unsigned long)*value*

Sets the value of the Control as an unsigned long. Returns *self*.

- setUpArrow:*aControl* **mirror:**(BOOL)*doMirror*

Sets the object that will have the keyboard focus after the user presses the up arrow. *aControl* must have the same Panel ancestor as *self*. If *doMirror* is YES, *aControl* will be set to have the down arrow return the focus here, unless the Control has another use for the down arrow (Text is a good example). Returns *self*.

- setUsesEvents:(BOOL)*enable*

Sets whether or not the Control responds to mouse and keyboard events. Returns *self*.

- takeDoubleValueFrom:*sender*

Makes a call to *sender* with the method **getDoubleValue**. Returns *self*.

- takeLongValueFrom:*sender*

Makes a call to *sender* with the method **getLongValue**. Returns *self*.

- takeStringValueFrom:*sender*

Makes a call to *sender* with the method **getStringValue**. Returns *self*.

- takeUnsignedLongValueFrom:*sender*

Makes a call to *sender* with the method **getUnsignedLongValue**. Returns *self*.

- (BOOL)**usesEvents**

Returns whether or not the Control responds to mouse or keyboard events.

Delegate Methods

- boundRectChanged:*sender*
from:(const rect *)*old*
to:(const rect *)*new*

Notifies the delegate that the bounding rectangle of the control has changed. *sender* is the control object. *old* and *new* are the before and after rectangles. Return value is irrelevant.

- boundRectWillChange:*sender*
from:(const rect *)*old*
to:(rect *)*new*

Notifies the delegate that the bounding rectangle of the control will be changed. *sender* is the control object. *old* and *new* are the before and after rectangles. The *new* value may be changed by this delegate, if desired. Note that interference in this manner could seriously disrupt the display of this control and should only be performed when it will result in a proper display. This method will be called after changes to the object have been requested internally. A change in *new* that makes it equal to *old* causes the rectangle not to change and will cancel calls to future delegates regarding this change. Return value is irrelevant.

- doubleValueChanged:*sender*
from:(double)*old*
to:(double)*new*
wasTaken:(BOOL)*taken*

Notifies the delegate that the value of the control has changed via a method other than one that changes the value by string. *sender* is the control object. *old* and *new* are the before and after values. *taken* states whether the value was changed by a **take*ValueFrom:** method (Interface Builder style). Return value is irrelevant.

- doubleValueWillChange:*sender*
from:(double)*old*
to:(double *)*new*
wasTaken:(BOOL)*taken*

Notifies the delegate that the value of the control will be changed via a method other than one that changes the value by string. *sender* is the control object. *old* and *new* are the before and after values. The *new* value may be changed by this delegate, if desired. A change in *new* that makes it equal to *old* causes the value not to change and will cancel calls to future delegates regarding this change. *taken* states whether the value was changed by a **take*ValueFrom:** method (Interface Builder style). Return value is irrelevant.

- keyPressed:*sender*
key:(const struct Event *)*event*
mouseat:(const point *)*where*

Called when a mouse press occurred within *sender*. *event* states which key is pressed and *where* indicates the point at which the key was pressed. Return value is irrelevant.

- keyReleased:*sender*
key:(const struct Event *)*event*
mouseat:(const point *)*where*

Called when a mouse release occurred within *sender*. *event* states which key is pressed and *where* indicates the point at which the key was pressed. Return value is irrelevant.

- mouseClicked:*sender*
buttons:(char)*buttonMask*
at:(const point *)*where*
num:(short)*num*

Called when a mouse click (successful press and release) occurred within *sender*. *buttonMask* states which buttons are pressed (BUTTON1 through BUTTON5), *where* indicates the point at which the button was pressed, and *num* indicates the number of clicks within the multiclick time limit. Return value is irrelevant.

- mouseEntered:*sender*
at:(const point *)*where*

Notifies the delegate that the mouse has entered the focus of the control. *sender* is the control object. *where* is the current point. Return value is irrelevant.

- mouseExited:*sender*
at:(const point *)*where*

Notifies the delegate that the mouse has left the focus of the control. *sender* is the control object. *where* is the last point the mouse was at before it left. Return value is irrelevant.

- mouseMoved:*sender*
from:(const point *)*old*
to:(const point *)*new*
buttons:(char)*buttonMask*

If mouse tracking is presently occurring, this notifies the delegate that the mouse has moved within the control. *sender* is the control object. *old* and *new* are the before and after points. *buttonMask* states which buttons are pressed (BUTTON1 through BUTTON5). Return value is irrelevant.

- mousePressed:*sender*
buttons:(char)*buttonMask*
at:(const point *)*where*

Called when a mouse press occurred within *sender*. *buttonMask* states which buttons are pressed (BUTTON1 through BUTTON5) and *where* indicates the point at which the button was pressed. Return value is irrelevant.

- mouseReleased:*sender*
buttons:(char)*buttonMask*
at:(const point *)*where*

Called when a mouse release occurred within *sender*. *buttonMask* states which buttons are pressed (BUTTON1 through BUTTON5) and *where* indicates the point at which the button was pressed. Return value is irrelevant.

- stringValueChanged:*sender*
from:(const char *)*old*
to:(const char *)*new*
wasTaken:(BOOL)*taken*

Notifies the delegate that the value of the control has changed via a method that changes the value by string. *sender* is the control object. *old* and *new* are the before and after values. *taken* states whether the value was changed by a **takeStringValueFrom:** method (Interface Builder style). Return value is irrelevant.

- stringValueWillChange:*sender*
from:(const char *)*old*
to:(char **)*new*
wasTaken:(BOOL)*taken*

Notifies the delegate that the value of the control will be changed via a method that changes the value by string. *sender* is the control object. *old* and *new* are the before and after values. The *new* value may be changed by this delegate, if desired. A change in *new* that makes it equal to *old* causes the value not to change and will cancel calls to future delegates regarding this change. *taken* states whether the value was changed by a **takeStringValueFrom:** method (Interface Builder style). Return value is irrelevant.

- viewRectChanged:*sender*
from:(const rect *)*old*
to:(const rect *)*new*

Notifies the delegate that the visible area of the control's rectangle has been changed. *sender* is the control object. *old* and *new* are the before and after rectangles. Return value is irrelevant.

- viewRectWillChange:*sender*
from:(const rect *)*old*
to:(rect *)*new*

Notifies the delegate that the visible area of the control's rectangle will be changed. *sender* is the control object. *old* and *new* are the before and after rectangles. The *new* value may be changed by this delegate, if desired. Note that interference in this manner could seriously disrupt the display of this control and should only be performed when it will result in a proper display. This method will be called after changes to the object have been requested internally. A change in *new* that makes it equal to *old* causes the rectangle not to change and will cancel calls to future delegates regarding this change. Return value is irrelevant.
To be written.

Class: Cursor

Inherits From: Object
Declared In: objx/Cursor.h

Class Description

The Cursor class is used to provide Images to be used as mouse cursors when over particular windows. This class handles the displaying of non-standard cursors, while X handles displaying of the standard ones. Generally a non-standard cursor can be defined in X terms as any pixmap cursor with more than one-bit depth or any cursor that has a size that the X server cannot handle. If and when X is capable of handling these cursors in the future, the Cursor class will take advantage of this ability.

Non-standard cursors are displayed by receiving a focusIn event from the owning window, making the real X cursor invisible. The Image used as the Cursor will be brought on-screen as a window at the point where the real cursor should be. All mouse tracking events while this cursor is to be displayed are brought here, and the Image's window is moved accordingly. Note that on slow systems or with large cursors, the motion of the Image may be jerky. If this is the case, a user may elect to use a standard cursor instead.

The Image used as a Cursor is copied into a new Image, so that any changes to the Image will not be reflected in the Cursor. (This is due to the fact that changing the cursors that X *can* handle won't be updated, so the ones it can't handle won't be either.) If you want to see the changes, simply re-install the Cursor into the proper windows.

Instance Variables

Image	**usedcursor**
Image	**usedmask**
Image	**image**
Image	**mask**
Point	**hotspot**
Point	**where**

usedcursor	The Cursor used when it's seen on-screen
usedmask	The mask for the Cursor. Not needed if the Image is shaped.
image	The Image object.
mask	The Image object set as a mask.
hotspot	The pointing area of the cursor.
where	The location of the cursor on the screen when the (non-standard) Cursor is being displayed

Method Types

Creating	- initImage:hotSpot:
Image management	- getImage - getMask - getHotSpot

	- setImage: - setMask: - setHotSpot:
Managing the Cursor on-screen	- cursorOn - cursorOff - cursorMoved: - isCursorHidden - hideCursor - showCursor

Class Methods

None

Instance Methods

To be written.

Class: Frame

Inherits From: View
Declared In: objx/Frame.h

Class Description

This class simply creates a flat or 3-D frame for a really cool appearance. Most objects use one of these internally.

Instance Variables

BOOL	**bezeled**
int	**thick**
Image	**inImage**
Image	**outImage**
Color	**inColor**
Color	**outColor**

bezeled	Is it 3-D or flat?
usedmask	Thickness of the border.
inImage	The Image used as the inward pointing border.
outImage	The Image used as the outward pointing border.
inColor	The Color used for the inward pointing border.
outColor	The Color used for the outward pointing border.

Method Types

Border appearance	- isBezeled - getBorderThickness - getBorderInColor - getBorderInImage - getBorderOutColor - getBorderOutImage
	- setBezeled: - setBorderThickness: - setBorderStyle: - setBorderInColor: - setBorderInImage: - setBorderOutColor: - setBorderOutImage:

Class Methods

None

Instance Methods

To be written.

Class: GSImage

Inherits From: Image
Declared In: objx/GSImage.h

Class Description

This class implements GhostScript in a manner accessible to the Image class tree. It can be used in two modes: one, (static mode) with backing store on or off, and the entire GS code set before drawing; or two, (interactive mode) with backing store on and each line of code can be drawn independently. Static mode is used for drawing objects with known drawing code or as a previewer, and interactive mode can be used to draw code as it comes without having to redraw the old code with each change. In interactive mode, a backing store is required because the GS code used to perform the drawing is not saved. The default mode is static.

The entire state of the interpreter is created for use with each object and is reset with each **drawself::** call while in static mode. In interactive mode, the state of the interpreter is never automatically reset.

Instance Variables

char *	**code**
BOOL	**staticmode**
?????	**anything_needed_for_the_GS_interpreter**

code	The code that the GS interpreter uses to redraw the image.
staticmode	Determines if the interpreter is in static or interactive mode.
?????	Interpreter specific variables. To be determined as the interpreter is written. The interpreter may end up as a seperate object.

Method Types

Static drawing code	- getGSCode - setGSCode: - addGSCode: - insertGSCode:beforeLine:
Interactive drawing code	- executeGSCode: - getExecStack - getDictStack - getGstateStack
Drawing mode	- isStaticMode - setStaticMode:
Displaying	- drawSelf::

Class Methods

None

Instance Methods

To be written.

Class: Image

Inherits From: Object
Declared In: objx/Image.h

Class Description

This is the head of the class tree that manages everything being displayed (but not the location of what's displayed). This class is never used directly, however. It is a frame for different drawing systems. The default one, XImage, handles drawing of images in plain X. An optional Image subclass is (will be, when I create it) GSImage. Regular GhostScript program code can be placed in that Image and it will be displayed in the same way as any other graphic. The Image class is designed to be extensible in such a manner.

This class tries to create a mode of device-independence by allowing a backing store even when the X server in question does not. It does this by using two pixmaps: one for the background pixmap/color and one for the foreground drawing. The foreground drawing is shaped by the Shape extension to allow the background to show through. Note that this method is used even when there is already a real backing store available, as it allows for some effects to be drawn that would otherwise be difficult.

Due to the fact that this forced backing store may be too inefficient for some X servers (or hardware), the feature can be shut off by telling the Image to draw only in a window. (The default is to draw in both, causing the drawing to be done in a pixmap and that pixmap to be the background pixmap for the foreground window.) When this happens, the Image will receive expose events (in the form of **drawSelf::** calls) and redraw the image as needed using the given drawing code. This drawing code will vary greatly from one Image type to another, so it is left up to those subclasses to define how that code is input.

Since Images contain subwindows and Image subclasses may contain further subwindows, it is unwise to attach windows onto Images manually, unless you know for certain that the class doesn't use subwindows (one example is PixImage: by definition, this class cannot create subwindows for itself). In most cases, you should create a window that is the parent of all of the windows you need.

Implementation note: I may decide to move the foreground pixmaps and windows to the XImage class because the GSImage doesn't need them, and others may not also. Those that do need them should implement them themselves for the sake of efficiency.

Instance Variables

Pixmap	**backpix**
Pixmap	**forepix**
Window	**backwind**
Window	**forewind**
BOOL	**usespix**
BOOL	**useswind**
BOOL	**usesbacking**
BOOL	**isModified**

backpix — The pixmap used as the background image. Not used if **usespix** is NO.

forepix	The pixmap used as the foreground image. Not used if **usespix** is NO.
backwind	The window used for displaying the background image. Not used if **useswind** is NO.
forewind	The window used for display the foreground image. Not used if **useswind** is NO.
usespix	Determines whether or not to draw into pixmaps.
useswind	Determines whether or not to draw into windows.
usesbacking	Determines if the windows should have backing store on. If NO, the image will have to redraw upon the receipt of expose events (**drawself::**'s).
isModified	A "dirty" flag for the Image. If set, the next **drawself::** should redraw the entire image, regardless of the rects.

Method Types

Output object	- usesPixmap - usesWindow - setUsingPixmap: - setUsingWindow: - getPixmap - getWindow
Backing store	- usesBackingStore - setUsingBackingStore:
Bounding	- getBoundRect - setBoundRect:
Storing	- writePixmapToFile:
Displaying	- drawSelf:: - isModified - setModified:

Class Methods

None

Instance Methods

To be written.

Class: Matrix

Inherits From: View
Declared In: objx/Matrix.h

Class Description

Holds and maintains rows and colums of objects. Automatically handles layouts when objects are added.

Instance Variables

int	**numRows**
int	**numCols**
BOOL	**autosizing**
int	**rowSpace**
int	**colSpace**
enum orientation	**orientation**

usedcursor	The Cursor used when it's seen on-screen
usedmask	The mask for the Cursor. Not needed if the Image is shaped.
image	The Image object.
mask	The Image object set as a mask.
hotspot	The pointing area of the cursor.
where	The location of the cursor on the screen when the (non-standard) Cursor is being displayed

Method Types

Size of matrix	- getNumRows - getNumCols - getNumCells - setNumRows: - setNumCols: - insertColBefore: - insertRowBefore:
Position of views	- getPointAtControl: - getControlAtPoint: - getViewAt:: - putView:at:: - addView:
View sizing	- isAutosizing - setAutosizing:
Space between views	- getSpaceBetweenRows - getSpaceBetweenCols - setSpaceBetweenRows: - setSpaceBetweenCols:

Matrix orientation	- getOrientation - setOrientation:

Class Methods

None

Instance Methods

To be written.

Class: Menu

Inherits From: Matrix
Declared In: objx/Menu.h

Class Description

Maintains a Matrix specially designed for menus, in that the menu can be have it layout in either horizontal or vertical direction, may be a part of an existing window (Motif/MS Windows style), may be at the top of the screen (Mac style) or may be in a separate window (NeXT style). Menus may also "split" themselves (be seen at two places on the screen at once); this class manages that.

Instance Variables

Panel	**menupanel**
Panel	**splitpanel**
id	**secondParent**

menupanel	The Panel object the menu uses when it is using a panel.
splitpanel	The Panel object the menu uses when it is split and uses a panel.
secondParent	The View that the menu used when it is split and not using a panel.

Method Types

Menu location	- isUsePanel - setUsePanel: - getPanel - setPanel:
Detaching the menu	- getSecondParent - setSecondParent:

Class Methods

None

Instance Methods

To be written.

Class: Panel

Inherits From: Control
Declared In: objx/Panel.h

Class Description

This class is used as the interaction between the interface and the Window Manager. Essentially it IS the Window class (I can't see the need for the Panel class and Window class to be separate).

This Panel class has the unique featue in X to be self managing. While many users would probably never use a self-managing window, it may be useful for the application programmer to have more accurate control over details such as window resizing. It also make it easier or the programmer not to have to worry about details such as icon size and color, differing demands of different window managers, and bad window managers misbehaving.

Instance Variables

char *	**title**
Font	**font**
BOOL	**hasTitlebar**
BOOL	**selfManaging**

title	The title of the window. May also need an icon title.
font	The font the title is displayed with. Not used if not self-managing.
hasTitlebar	Determines whether or not a titlebar exists at all. selfManaging is irrelevant.
selfManaging	Determines whether or not the window is self managing.

Method Types

Displaying title	- getXFont - getXTitle - getTitleAttachments - hasTitleBar - setXFont: - setXTitle: - setTitleAttachments - setHasTitleBar:
Internal objects	- getMaxButton - getMinButton - getCloseButton - getSysmenuButton - getResizeControl - setMaxButton: - setMinButton: - setCloseButton: - setSysmenuButton:

	- setResizeControl:
Window management	- isSelfManaging - setSelfManaging
Text placement	- getTextAlignment - setTextAlignment:
Border appearance	- getBorderThickness - getBorderStyle - getBorderInColor - getBorderInImage - getBorderOutColor - getBorderOutImage - setBorderThickness: - setBorderStyle: - setBorderInColor: - setBorderInImage: - setBorderOutColor: - setBorderOutImage:

Class Methods

None

Instance Methods

To be written.

Class: PixImage

Inherits From: Image
Declared In: objx/PixImage.h

Class Description

The PixImage class is designed to receive pre-made pixmaps for use by methods requiring Images.

Instance Variables

Pixmap **pixmap**

pixmap — The pixmap set by the application. ***Implementation note:*** May be able to use the pixmap variable from the Image class.

Method Types

Creating and freeing	- init - free
Output object	- usesPixmap - setUsingPixmap:
Bounding	- getBoundRect - setBoundRect:
Setting the Pixmap	- getPixmap - setPixmap: - readPixmapFromFile:

Class Methods

None

Instance Methods

To be written.

Class: Scrollbar

Inherits From: Control
Declared In: objx/Scrollbar.h

Class Description

This class manages the Scrollbar and it's different regions: the buttons, slider, and well. By default, the Scrollbar creates all parts. If you don't want any particular button, set it to nil, and it will be destroyed. If you set your own buttons, resetting to nil will destroy them, resetting to NULL will just no longer use them. (Is this a good idea? How else to manage auto creation/destruction while still being able to set the buttons manually?)

Instance Variables

Image	**knob**
enum orientation	**orientation**
double	**min**
double	**max**
int	**sliderminsize**
Button	**upleftbutton**
Button	**downrightbutton**
Button	**pageupleftbutton**
Button	**pagedownrightbutton**

Determines whether or not the ScrollView's subview will be scrolled automatically.
Show the horizontal scrollbar always, only when needed, or never?
Show the vertical scrollbar always, only when needed, or never?

Method Types

Knob	- getKnobImage - setKnobImage: - getSliderMinSize - setSliderMinSize:
Orientation	- getOrientation - setOrientation:
Value	- getMinLongValue - getMinULongValue - getMinDoubleValue - getMaxLongValue - getMaxULongValue - getMaxDoubleValue - setMinLongValue: - setMinULongValue: - setMinDoubleValue: - setMaxLongValue:

	- setMaxULongValue: - setMaxDoubleValue:
Buttons	- getLineButtonA - getLineButtonB - getPageButtonA - getPageButtonB
	- setLineButtonA: - setLineButtonB: - setPageButtonA: - setPageButtonB:

Class Methods

None

Instance Methods

To be written.

Class: ScrollView

Inherits From: View
Declared In: objx/ScrollView.h

Class Description

This class handles all aspects of automatic or manual scrolling of a large area.

Instance Variables

BOOL	**autoscroll**
enum scrollwhen	**hscrollwhen**
enum scrollwhen	**vscrollwhen**

autoscroll	Determines whether or not the ScrollView's subview will be scrolled automatically.
hscrollwhen	Show the horizontal scrollbar always, only when needed, or never?
vscrollwhen	Show the vertical scrollbar always, only when needed, or never?

Method Types

Subobjects	- getHScrollbar - getVScrollbar - getSubView - setHScrollbar: - setVScrollbar: - setSubView:
Automation	- getAutoscroll - setAutoscroll:
Scrollbar visibility	- getWhenHScrollbarShown - getWhenVScrollbarShown - showHScrollbarWhen: - showVScrollbarWhen:

Class Methods

None

Instance Methods

To be written.

Class: View

Inherits From: Object
Declared In: objx/View.h

Class Description

This is the main class in this library. Manages the existance and location of subviews and images, parentry, event passing, view clipping and cursor windows.

Instance Variables

id *	**subviews**
id	**parent**
BOOL	**passEvent**
Rect	**clipRect**

subviews	A list of Views that share this View as their parent. The View class merely contains them and redraws them at their requested locations. It is up to any subclasses to control the desired positioning of these objects.
parent	The parent View of this View.
passEvent	Determines whether or not events given to this view are passed to it's children.
clipRect	The clipping rectangle of the view.

Method Types

Ancestry	+ getRootView - getParent - setParent:
Responding to events	- isPassEventToChild - setPassEventToChild: - getChildAtPoint:
Image location	- getAttachments - setAttachments: - getBoundRect - setBoundRect: - getClipRect - setClipRect
Cursor management	- getCursorRects - setCursor:rect:
Border appearance	- getBackgroundColor - getBackgroundImage - getForegroundColor - getForegroundImage

	- setBackgroundColor: - setBackgroundImage: - setForegroundColor: - setForegroundImage:
Displaying	- drawSelf::

Class Methods

None

Instance Methods

To be written.

Class: Text

Inherits From: Control
Declared In: objx/Text.h

Class Description

Still working on these.

Instance Variables

Method Types

Class Methods

Instance Methods

Class: TextField

Inherits From: Control
Declared In: objx/TextField.h

Class Description

Instance Variables

Method Types

Class Methods

Instance Methods

Class: XImage

Inherits From: Image
Declared In: objx/XImage.h

Class Description

The Image class for drawing straight in Xlib.

Implementation note: I'm still not sure in what format code should be in... maybe a typedstream like data structure consisting of pointers to X functions and their arguments, or maybe a function/method call?

Instance Variables

? **drawingCode**

drawingCode The commands used to redraw the image when necessary.

Method Types

Drawing Code	- getDrawingCode - setDrawingCode: - addDrawingCode:
Displaying	- drawSelf::

Class Methods

None

Instance Methods

To be written.