Project Title: ***EuroBridge***

Project Number: **RACE 2008**

Deliverable Type: **I (Internal)**

Title Of Deliverable: ***EuroBridge W idget Set***

(Version 4.0)

CEC Deliverable Number: **Dxx**

Contractual Date Of Delivery: **-**

Actual Date Of Delivery: **March 27, 1996**

Produced By Workpackage: **WP2**

Nature Of Deliverable: **R (Report)**

Authors: **Markku Savela (VTT)**

Abstract:

This document is the programmers manual for the EuroBridge Widget Set.

Keywords:

EuroBridge, Widget Set

Title	Related Time Period (or Date)	CEC Deliverable
EuroBridge Widget Set		Dxx

Prepared by	Document Date	Filename
Markku Savela		Toolkit.book

Checked - Signature	- Date	Approved - Signature	- Date

Executive Summary

EuroBridge widget set is a design and implementation of a toolkit for displaying, presenting and printing the information which is exchanged between communicating multimedia applications, for example mail user agent or client programs for distributed information retrieval. The toolkit is based on standard X Window System and built on top of Xt library. It defines a set of basic data presentation widgets which hold the information imported in one of the defined interchange formats. The widgets take care of the details of transforming this information into a visible form on a window on the work station screen.

The widget set is independent of the user interface issues, it is designed to be supplementary to the existing widget sets (for example MOTIF Widgets from OSF or Athena Widgets from X11 distribution).

This document consists of the specification of the programming interface to the widget set. A sample implementation of the EuroBridge Widget Set (Xew) is distributed in source form.

The sample implementation of the widget set is freely usable. First because it relies heavily on other freely usable software packages, and second to accelerate the development of multimedia capabilities in Unix environment. The widget set offers a cheap and easy to use starting point for many experimenters in this area. The widget set is not intended to be a multimedia authoring application, it is intended to be a basis for widely distributed applications which have a simple need to present communicated multimedia information generated by multimedia authoring and other similar applications.

This release (Version 4.0) of the sample implementation supports text, image, audio and video. The Text widget supports directly the ISO 8859-1 character set, and through the code extension techniques defined in ISO 2022 it supports any character sets available in the X server (for example Japanese Character Set JIS C 6226-1983, Korean Standard KSC 5601-1987). Also, a subset of controls from ISO 6429 is supported, most important of these being the Select Graphic Rendition (SGR), which gives the possibility to use different point sizes, colours and renditions within text content. The raster widget can import TIFF, JPEG/JIF, GIF and PBM encoded images. The audio widget supports 8kHz u-law encoding on SunOS/SPARC architecture. The video widget imports MPEG-1 streams (implementation is slow).

Copyright Notice

Table of contents

Toolkit Widgets

1. Introduction

This document was written as a part of Work Package 2 of the EuroBridge (RACE 2008) project.

The purpose of this document is to specify and describe a sample implementation of a basic multimedia widget set for X Window system to be used in presenting the information formats specified in the companion document *Interchange Formats*[1].

Each main category (Text, Images, Graphics, Audio, Video) will have own widget class. All widgets have the common characteristic that they will import the content description as an octet stream from any of the following sources

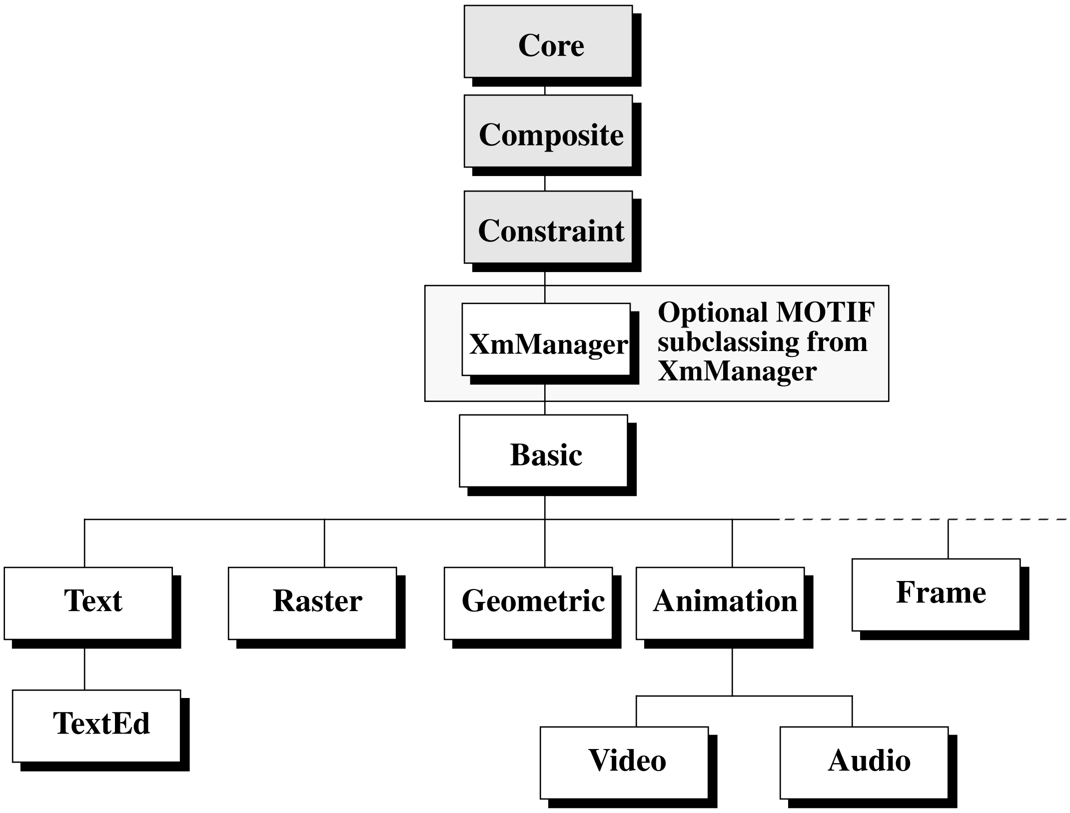
- a named file
- open file descriptor
- a shell command standard output
- pointer to memory

Full information of the imported content is preserved within the widget in internal format and all export operations (cut and paste) operate on this internal presentation.

This document specifies widgets only for the five basic interchange formats, which are Text, Raster image, Geometric image, Audio and Video. These widgets will only visualise the content without making any decisions about user interface issues. Deciding about the user interface is left to the applications and maybe some higher level widgets based on these.

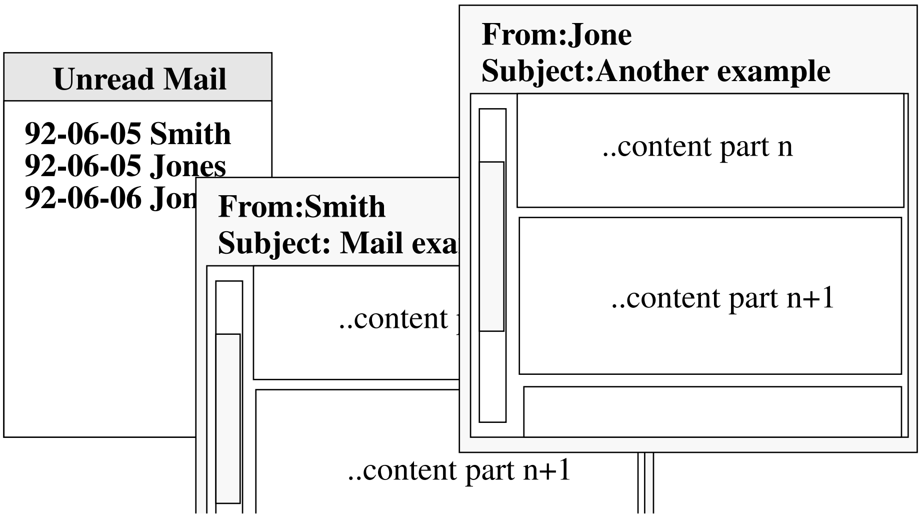
Currently, Text, Raster, Audio and Video have an implementation. There is also a geometry manager widget Frame.

The following picture shows the class hierachy of the basic categories.

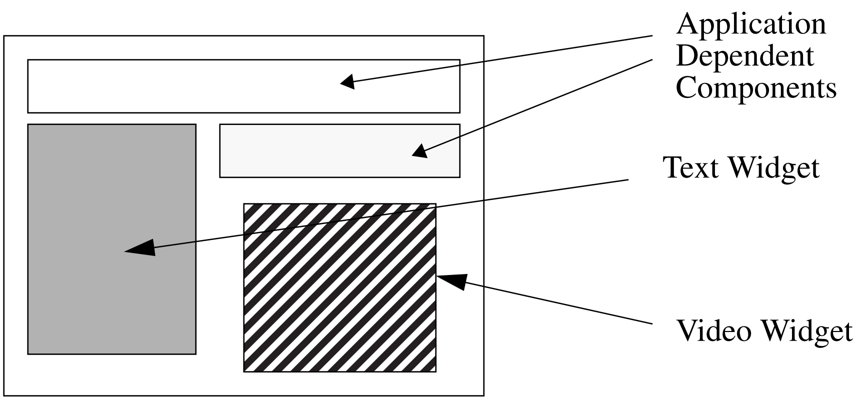


Basic widgets are intended to be used within applications that have a need to present the chosen interchange formats to the user. Examples are

- Mail User agent. One part of UA could be window which will be used to present the currently active mail message. The basic widgets would be used to present each individual content part. The UA would choose the appropriate widget type for each content type from the information available in the mail message.



- Document/information retrieval application over the network would probably look very much like the above mail user agent, having multiple windows containing various selection information, control panels etc., and then one or more document windows showing the actual information content.
- Sign Language Application can use the video widget for the video sequences and text widget for textual information along whatever application dependent components it needs.



- compound document viewer is a complex combination of basic widgets that are laid out into pages according to the layout rules specified within the document.

EuroBridge Widgets will follow the conventions of interclient communication as defined by X Consortium in *Inter-Client Communications Manual* (ICCCM). Each widget class will support selections[1], which enable exchange of content information between widgets. The following table contains a tentative list of the known standard target types suggested in ICCCM that might be supported by each widget class.

Table 1: Some target types suggested in ICCCM

Widget	Import	Export
Text	STRING, COMPOUND_TEXT	STRING, COMPOUND_TEXT
Image	PIXMAP?	PIXMAP?
Graphic	?	BITMAP?
Video	PIXMAP?	PIXMAP?
Audio	?	?
Document	ODIF?,...	ODIF?,...
Hypermedia	?	?

As can be seen, the standard list is not very complete and lacks functionality in several important areas. In addition to standard targets, some additional privately defined targets between EuroBridge Widgets need to be defined or chosen. Each of the recommended interchange formats could be one additional target that EuroBridge defines.

Following ICCCM conventions allow EuroBridge applications to exchange information through cut and paste operations with other X Window applications, and even with MS Windows applications, when X Server on PC/MS-DOS environment is used.

1. However, in the current release (Xew 4.0), only Text supports selections. Even after 4 years of the project, the standardisation of the multimedia target types for X Window system has not progressed much.

2. Basic Widget

Application header file	<X11/Xew/Basic.h>
Class header file	<X11/Xew/BasicP.h>
Class	xeBasicWidgetClass
Class Name	XeBasic
SuperClass	Constraint

The Basic Widget class alone does not present any content information, but is a superclass for all other EuroBridge Widgets. The Basic Widget defines the common resources for all EuroBridge Widgets.

Name	Class	Type	Notes[a]	Default Value
activateCallback	ActivateCallback	Pointer		NULL
colormapInstall	ColormapInstall	Boolean		False
colormapUse	ColormapUse	XeColormapUse		XeColormapUse_SHARED
colorMode	ColorMode	XeColorMode		XeColorMode_COLOR
colorQuantize	ColorQuantize	XeColorQuantize		XeColorQuantize_FAST
contentFile	ContentFile	String	I	NULL
contentFilter	ContentFilter	String	I	NULL
contentLength	ContentLength	long	I	Length of the content or end of file
contentLoaded	ContentLoaded	Boolean	I	False
contentOffset	ContentOffset	long	I	0
contentStream	ContentStream	FILE*	I	NULL
contentString	ContentString	String	I	NULL
cursor	Cursor	Cursor		None
dither	Dither	XeDither		XeDither_FS4
exportCallback	ExportCallback	Pointer		NULL
exposeCallback	ExposeCallback	Pointer		NULL
horizontal	FrameDimension	XeFrameDimension	(N)	XeFrameDimension_MAX,0,0
foreground	Foreground	Pixel		XtForegroundPixel
importCallback	ImportCallback	Pointer		NULL
maxColors	MaxColors	int		65535
mirrorImage	MirrorImage	Boolean		False
notifyCallback	NotifyCallBack	Pointer		NULL
resize	Resize	Boolean		True
rotation	Rotation	int		0
scaling	Scaling	XeScaling		XeScaling_WIDTH
userData	UserData	Pointer		NULL
useShm	UseShm	Boolean		False
vertical	FrameDimension	XeFrameDimension	(N)	XeFrameDimension_MAX,0,0
visual	Visual	Visual *		NULL
xTranslation	XTranslation	int		0
yTranslation	YTranslation	int		0

a. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial implementation in some widget classes. "I" in this column indicates resources in which changes will become effective only when the content is loaded/imported into the widget (that is, when **contentLoaded** resource is set to False).

Each basic widget covers a rectangular area and defines the resources for the content information. The content can be defined in three different ways: as a filename, as an open file descriptor or as a pointer to memory area. Passing the content via open file descriptor is important for the real time widgets (audio and video) and allows them to display continuously from network connections.

2.1 Defined resources

activateCallback (`XeCR_ACTIVATE`)
is a generic callback list for miscellaneous "activate type" events generated by the activate action. This callback is left for the application use (a normal use would be to bind the activate action to some application specific event). The reason field in the callback data has the value `XeCR_ACTIVATE`.

```
typedef struct XeActivateCallbackData
  {
      int reason;     /* == XeCR_ACTIVATE == */
      XEvent *event;  /* event triggering the callback */
      String *params;/* Point to args of 'notity(args)' */
      Cardinal num_params;/* Num arguments in 'args' */
  } XeActivateCallbackData;
```

colormapInstall
The proper way to install colormaps is to use **XtSetWMColormapWindows**, but some window managers don't seem to support it. If **colormapInstall** resource is set True, the widget will use direct **XInstallColormap/XUninstallColormap** calls instead. The default value is False.

colormapUse controls the strategy of how colormaps are used. This concerns currently only Raster and Video widgets (see also **maxColors**), and

XeColormapUse	
Enumeration value (name)	**Converted from String**
`XeColormapUse_SHARED`	shared
`XeColormapUse_SHAREDOWN`	sharedown
`XeColormapUse_OPTIONAL`	optional
`XeColormapUse_PRIVATE`	private

only with PseudoColor, DirectColor or GrayScale visuals.

`XeColormapUse_SHARED`

The colours for the image are allocated from the colormap that is defined in core.colormap resource. If not all colours could be allocated, the remaining colours are chosen from the colormap without allocating them. (This means that the image will use read-write colours allocated somewhere else and, which may change and disturb the image).

`XeColormapUse_SHAREDOWN`

The colours for the image are allocated like in SHARED, but the colours that could not be allocated will be approximated from the allocated colours instead of looking at all of the colormap. (This means that the image has less colours from which to choose, but it will not be affected by changes to the shared colormap elsewhere).

`XeColormapUse_OPTIONAL`

First, the widget attempts to allocate all required colours from the colormap that is defined in the core.colormap resource. If this allocation fails, then a private colormap for this widget is used.

`XeColormapUse_PRIVATE`

The widget always uses a private colormap.

colorMode can be used to request displaying of the content with less colour capabilities that the actual visual would allow. If this resource requests more capabilities than the visual allows, widgets will automatically downgrade to the level specified by the visual. Used by Raster and Video widgets.

XeColorMode	
Enumeration value (name)	**Converted from String**
`XeColorMode_NONE (not used)`	
`XeColorMode_MONO`	mono
`XeColorMode_GRAY`	gray
`XeColorMode_COLOR`	color

`XeColorMode_MONO`

Use only foreground and background colours.

`XeColorMode_GRAY`

Use only shades of gray.

`XeColorMode_COLOR`

Use colours, if possible.

colorQuantize controls the algorithm which is used in converting from 24 bit colour format into palette colour format. This resource is only used if the selected visual is PseudoColor or StaticColor. Used by Raster and Video widgets.

XeColorQuantize	
Enumeration value (name)	**Converted from String**
`XeColorQuantize_HECKBERT`	heckbert
`XeColorQuantize_FAST`	fast
`XeColorQuantize_PPMQUANT`	ppmquant
`XeColorQuantize_JPEG`	jpeg

`XeColorQuantize_HECKBERT`
Heckbert's median cut.

`XeColorQuantize_FAST`
Floyd-Sternberg dithering.

`XeColorQuantize_PPMQUANT`
This will use the ppmquant algorithm from the PBMPLUS package (may not work properly, seems that this does not produce good results with few colours)

`XeColorQuantize_JPEG`
Request colour quantification to be made internally in IJG while decoding the JPEG image.

This is effective only at a point when the JPEG image is decoded. If specified for other than JPEG, the default XeColorQuantize_FAST is used instead.

contentFile A filename identifying the file that contains octet stream for the content. This octet stream must represent the content of the widget in one of the supported interchange formats.

Since release 3.0, the widget will use a private copy of the string.

contentFilter A system command whose standard output will form the content octet stream of the widget. The filter is opened with a `popen(filter,"r")` call.

Since release 3.0, the widget will use a private copy of the string.

contentFormat

will specify the interchange format which is used in the content. The tentative list of formats is:

XeContentFormat		
Enumeration value (name)	**Converted from String**	**Comment**
`XeContentFormat_UNKNOWN`	unknown	(if nothing specified)
`XeContentFormat_ISO2022`	iso2022	Text stream (ISO 2022)
`XeContentFormat_TIFF`	tiff	Tagged Image File Format
`XeContentFormat_JPEG`	jpeg	Joint Photographic Experts Group
`XeContentFormat_GIF`	gif	Graphics Interchange Format
`XeContentFormat_PBM`	pbm	Portable Bitmap
`XeContentFormat_RLE`	rle	Utah Raster Toolkit
`XeContentFormat_CGM`	cgm	Computer Graphics Metafile
`XeContentFormat_H261`	h261	Video stream (CCITT H.261)
`XeContentFormat_MPEG`	mpeg	Video Stream
`XeContentFormat_AUDIO`	audio	Basic u-law 8kHz audio
`XeContentFormat_ODIF`	odif	Open Document Interchange Format

If **contentFormat** is `XeContentFormat_UNKNOWN`, the format specific widget subclass will choose the default value appropriate for that widget class.

contentStream

An open file descriptor from which the octet stream representing the content will be read.

*The widget does not make a private copy of this resource. The value must be valid at widget creation time and every time the **contentLoaded** is set **False.***

contentString A pointer to memory area containing the octet stream representing the content.

*The widget does not make a private copy of this resource. The value must be valid at widget creation time and every time the **contentLoaded** is set **False.***

*Only one of the resources **contentFile**, **contentFilter**, **contentStream** or **contentString** will be effective. The one used will be the first non-NULL content in order **contentString**, **contentStream**, **contentFile**,*

***contentFilter**. For example, if an application has used **contentString** and then later wants to use **contentFile**, it must reset **contentString** to NULL to enable use of **contentFile**.*

contentLength
The length of the included content in octets, if non-zero. When zero, all of the content from offset position to the end is included. The logic of finding the end of the content depends on the method of content is being provided. For **contentFile**, **contentFilter** and **contentStream** the end is at EndOfFile; for **contentString** the value defaults to ZERO.

*The **contentLength** is not intended to be used as a method of querying the actual length of the octet stream presentation of the content. This resource is input only. This is to avoid problems with **contentFile** and **contentStream** implementations. With them it may not always be easy to find the real octet length in all environments.*

contentLoaded
The value of this resource is examined in intrinsic *initialize* and *setvalues* methods. If the value at that point is False, the widget will examine the other content specifying resources (**contentFile**, **contentFilter**, **contentStream**, **contentString** etc.) and (re)load the internal content from them. After load has been completed the contentLoaded will be set to True. If the value of this resource is True at *initialize* or *setvalues*, the internal content of the widget is not changed.

contentOffset The number of octets to skip from the content before the actual octets belonging to the content begin.

*The **contentLength** resource counts from the offset point, the octets skipped by the **contentOffset** are not included in **contentLength**.*

This resource might be useful when a single file contains multiple contents. This enables the use of same file without a need to copy the content explicitly into memory or to temporary files.

cursor The image that will be displayed as the pointer cursor whenever it is in this widget.

dither specifies whether image dithering for display is done or not. Affects rendering of some Raster and Video images, but is not consistently implemented.

XeDither	
Enumeration value (name)	**Converted from string**
`XeDither_NONE`	none
`XeDither_FS4`	fs4

`XeDither_NONE`disables dithering, when applicable (for some Raster/Video).

`XeDither_FS4`enables a version of Floyd-Steinberg dithering, when applicable (for some Raster/Video)

exportCallback (`XeCR_EXPORT`)
is called from the widget `ConvertSelection` method. This callback gives the application a chance add own export targets and conversions. The reason field in the callback data has the value `XeCR_EXPORT`.

```
typedef struct XeExportCallbackData
    {
        int reason;             /* XeCR_EXPORT */
        Time time;              /* Timestamp */
        Atom selection;         /* PRIMARY, ... etc/
        Atom target;            /* ..type */
        Atom type;              /* ..of the value */
        XtPointer value;        /* address of the value */
        unsigned long length;   /* ..of the value */
        int format;             /* unitsize (8,16,32)*/
    } XeExportCallbackData;
```

If the target field has value `XA_TARGETS`, then the fields type, value, length and format are already prefilled with the targets the widget is prepared to deal. An application can decide to remove or add own private targets. For any other targets, the fields type, value, length and format have ZERO value and an application can supply own converted value if it so decides. When returning from the callback, if `length` *and* `format` have non-zero values, the widget assumes the callback has converted the selection.

If the callback does not want to do anything with the conversion, it can just return. In that case, the widget code will use internal default conversion.

exposeCallback (`XeCR_EXPOSE`)
is called from the widgets expose method *after* all the widget specific

expose handling has been completed. The `reason` field in the callback data has the value `XeCR_EXPOSE`.

```
typedef struct XeExposeCallbackData
    {
        int reason;             /* XeCR_EXPOSE */
        XExposeEvent *event;    /* ..from expose method */
        Region region;          /* ..from expose method */
    } XeExposeCallbackData;
```

Application should be prepared for a case where the event pointer is NULL. This indicates that the expose event is not really from the server, but is generated internally within the widget (currently at least TextEd generates such events).

With **exposeCallback** (and **notifyCallback**) the application can use any widget as "canvas" or "drawing area", where the widget provides the fixed background (Basic widget provides blank background).

foreground determines the foreground colour for the widgets to use. The actual use of this depends on the widget class. In text widget this colour is the default foreground colour for the text (and the core **background** resource is the background colour for the text).

horizontal, **vertical** (**FrameDimension**)
can be used to explicitly specify and/or constrain the preferred geometry of the widget.

Note, that the resource name is the same as the one used as equivalent constraint resource in other widgets (Text, Frame). If the parent is such widget, setting this resource will set both the constraint and the widget resource. The semantics of XeFrameDimension is such that this situation is perfectly normal and valid.

```
typedef struct XeFrameDimension
    {
        int value;
        int min, max;
    } XeFrameDimension;
```

`value > 0` if greater than zero, it specifies the preferred size of this widget in the specified dimension (horizontal or vertical).

`value <= 0` if less or equal to zero, the preferred dimension is dynamically computed and depends on the widget class (`XeFrameDimension_MAX`, `XeFrameDimension_RULE_A`, `XeFrameDimension_RULE_B`). In such case the preferred size is usually the smallest possible size to contain the content.

`min`, `max` In all cases, if either field `min` and/or `max` have positive values, they will further constrain the dimension. Zero value in the field means "no limit". (*Do not use negative values, left for future use*)

importCallback (`XeCR_IMPORT`)
is called from the widget `SelectionReceived` method (see the description of **XtSelectionCallbackProc**) and gives the application a chance to process the received data. The reason field in the callback data has the value `XeCR_IMPORT`.

```
typedef struct XeImportCallbackData
    {
        int reason;             /* XeCR_IMPORT */
        Atom selection;         /* PRIMARY etc..*/
        Atom type;              /* Type of the value */
        XtPointer value;        /* returned value */
        unsigned long length;   /* Length of the value */
        int format;             /* Bitsize (8, 16, 32) */
    } XeImportCallbackData;
```

On entry to the callback the callback data contains the data to be received. If the application does not process this self, it can just return without changing anything. In this case widget will self do the default importing.

If application processed the received data in the callback, it must set the length field to ZERO (it can also just modify the content in place, and update the length field, if that is easier). If value field is left non-NULL, the widget will release the memory allocated, but application can also grab the value off and return NULL instead. In this case, it is the responsibility of the application to release the memory at some point.

maxColors limits the number of colours that will be allocated from the colormap (see **colormapUse**). Restricting colours that may be allocated is advisable in PseudoColor, DirectColor or GrayScale visuals, if the purpose is to display multiple images at same time. If no restriction is made, the first multicoloured image will allocate all colours and the remaining images just have to use only existing colours, they cannot add any of their own to the colormap.

The maxColors resource has a special interpretation when the value is less than 2.

maxColors	
Value	**Interpretation**
`0`	Represent image using dithering into two levels. The current algorithm is Floyd-Steinberg dithering.
`1`	(Reserved for future definition, currently acts like value was 0)
`> 1`	Allocate at most this many colours for the image.

mirrorImage requests the content to be shown as mirror image. Only supported by the Raster and Video widgets. (see also **rotation**).

notifyCallback

is a generic callback list for miscellaneous “notify type” events that the widget needs to inform the application about. *All callbacks* generated by the Xe widgets pass as a callback_data a structure that has the following basic format:

```
struct
    {
        int reason;     /* Reason code defined by XeCR */
        ...             /* Other fields depending reason*/
    }
```

The following table lists *all known* (not only notifyCallback) reason codes:

XeCR	
Enumeration value (name)	**Description**
`XeCR_UNKNOWN`	Unspecified (not used)
`XeCR_NOTIFY`	Generic (unspecific) action notify report
`XeCR_NOTIFY_AREA`	Generic Area of interest report
`XeCR_EXPOSE`	Generic Expose callback
`XeCR_MODIFY_TEXT`	TextEd callback (see TextEd)
`XeCR_LAYOUT_CHANGE_TAG`	Text Layout callback (see Text)
`XeCR_LAYOUT_BEGIN_LINE`	Text Layout callback (see Text)
`XeCR_ANIMATION_DONE`	Animation callback (see Animation)
`XeCR_ANIMATION_FRAME`	Animation callback (see Animation)
`XeCR_EXPORT`	Universal Convert Selection callback
`XeCR_IMPORT`	Universal Receive Selection Callback
`XeCR_FRAME_OVERFLOW`	Frame layout callback (see Frame)
`XeCR_RASTER_IMAGE`	Raster loaded callback (see Raster)

XeCR	
Enumeration value (name)	**Description**
`XeCR_ACTIVATE`	Activate callback

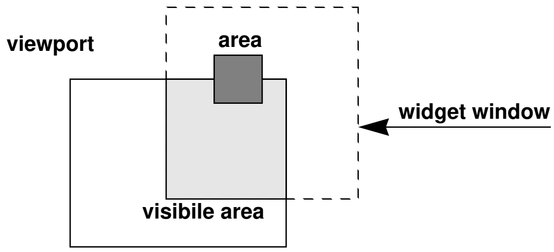
The following two notify callbacks occur via **notifyCallback** and can be used by all widget classes (the detailed method of use may vary from class to class, but the generic idea will be the same).

`XeCR_NOTIFY` This callback is generated by `notify()` action. The widget set does not associate any internal semantics with this action and it can be used for application needs. Some widget classes may have some default binding to this action. The callback_data structure is:

```
typedef struct XeNotifyCallbackData
    {
        int reason;             /* XeCR_NOTIFY */
        XEvent *event;          /* Triggering Event */
        String *params;         /* Args in notify(args) */
        Cardinal num_params;    /* Number of args */
    } XeNotifyCallbackData;
```

`XeCR_NOTIFY_AREA`

is a notifyCallback intended to be used in scrolling or panning the widget content within viewport, porthole or other similar constructs. The callback data contains two rectangles expressed in the coordinate space of the widget: an area that the widget wishes to be visible (like the cursor position in TextEd widget) and the current visible window:



It is up to the application to decide how to comply with the request.

```
typedef struct XeNotifyAreaCallbackData
    {
        int reason;             /* XeCR_NOTIFY_AREA */
        XRectangle area;        /* Area of interest */
        XRectangle visible;     /* Visible area of window */
    } XeNotifyAreaCallbackData;
```

resize *after* the effects of **scaling** have been computed, the **resize** resource determines whether the widget requests resizing the window to the exact computed size (True) or not (False).

rotation indicates the number of degrees the widget content is to be rotated clockwise from the normal position. Currently only 0, 90, 180, 270 degrees are supported, and only for Raster and Video widgets. (see also **mirrorImage**).

scaling determines how the visualized size of the widget content is related to the "real size" of the content (see also **resize**). The exact interpretation of what "real size" means is dependent on the widget class. The possible values are defined by the enumeration values:

XeScaling	
Enumeration value (name)	**Converted from String**
`XeScaling_NONE`	none
`XeScaling_WIDTH`	width
`XeScaling_HEIGHT`	height
`XeScaling_MAXASPECT`	maxaspect
`XeScaling_FITAREA`	fitarea

`XeScaling_NONE`

the visualized size is determined from the content, if possible. This mode is a request to display the content in real, resolution independent size. If the interchange format of the content doesn't specify any real size, some format specific defaults are assumed.

`XeScaling_WIDTH`

the content is scaled horizontally to match the window width and the height is computed such that the aspect ratio is preserved. Note that the computed height may exceed or be less than the current window height.

If the window width is unspecified (0), the effect is as if `XeScaling_NONE` was selected.

`XeScaling_HEIGHT`

the content is scaled vertically to match the window height and the width is computed such that the aspect ratio is preserved. Note that the computed width may exceed or be less than the current window width.

If the window height is unspecified (0), the effect is as if `XeScaling_NONE` was selected.

`XeScaling_MAXASPECT`
the content is scaled such that it has the maximum size that fits the window rectangle and still preserves the aspect ratio.

If either window width or height is unspecified (0), the effect is as if either `XeScaling_HEIGHT` or `XeScaling_WIDTH` was selected. If both are undefined, the effect is as if `XeScaling_NONE` was selected.

`XeScaling_FITAREA`
the content is scaled such that both width and height match the window rectangle. This scaling method may change the aspect ratio of the visualized content.

If either window width or height is unspecified (0), the effect is as if either `XeScaling_HEIGHT` or `XeScaling_WIDTH` was selected. If both are undefined, the effect is as if `XeScaling_NONE` was selected.

useShm when True, the widget will attempt to utilize the MIT-SHM extension of the X server, if possible. Currently effective only for Raster and Video Widgets.

vertical see **horizontal**.

visual if defined, will cause the widget to have this specific visual (assuming the server supports it). This resource is effective only if set before realizing the widget. If undefined, the visual is inherited from the parent.

xTranslation, **yTranslation**
resources translate the content origin to the widget window origin. The default (0,0) has content placed on window origin.

2.2 Use of resources from superclass(es)

borderWidth This Core resource is forced to 0 in the initialize of Basic widget.

2.3 Translations and actions

The following actions are defined for the Basic class:

Miscellaneous

`activate([parameter[,parameter ...]])`

call the Basic **activateCallback** with reasone `XeCR_ACTIVATE` and pass the parameter strings to the callback in the XeActivateCallbackData structure.

`notify([parameter[,parameter ...]])`

call the Basic **notifyCallback** with reason `XeCR_NOTIFY` and pass the parameter strings.

`enter-window(), leave-window()`

The actions take care of the colormap switching, if the widget has been given a private colormap. If the resouce **colormapInstall** has value True, the functions XInstallColormap and XUninstallColormap are used. If False, the XtSetWMColormapWindows is used.

The default event bindings for the XeBasic widget are:

```
static char defaultTranslations[] =
        "\
        <Btn1Down>,<Btn1Up>:    notify()\n\
        <EnterWindow>:          enter-window()\n\
        <LeaveWindow>:          leave-window()\n\
        ";
```

2.4 Installed converters

The Basic class installs the following resource converters:

from_type	to_type	Description
XtRString	XtRPixel	Replaces the default Xt converter with a version that uses internal **XeAllocColors** (convert never fails).
	XtRXeColomapUse	Convert string to **XeColormapUse** value
	XtRXeColorMode	Convert string to **XeColorMode** value
	XtRXeDither	Convert string to **XeDither** value
	XtRXeColorQuantize	Convert string to **XeColorQuantize** value
	XtRXeScaling	Convert string to **XeScaling** value
	XtRXeContentFormat	Convert string to **XeContentFormat** value

3. Text Widget

Application header file	<X11/Xew/Text.h>
Class header file	<X11/Xew/TextP.h>
Class	xeTextWidgetClass
Class Name	XeText
SuperClass	XeBasic

A Text Widget is a rectangular area, that holds an ISO 2022 conformant octet stream, usually representing a paragraph of text. The widget will attempt to present this octet stream in area as well as it can. For unrecognized character sets a special fallback visualization method is used. This will not produce the real character shapes, but still preserves the original information for the purposes of cut and copy from the widget. At minimum, this widget will fully support ISO 8859-1 character set.

Name	Class	Type	Notes[a]	Default Value
alignment	Alignment	Alignment		XeAlignment_START
columnHeight	ColumnHeight	int		0
columns	Columns	int		1
columnSeparation	ColumnSeparation	int		0
columnWidth	ColumnWidth	int		0
exportFormat	ExportFormat	XeTextExport		XeTextExport_ODIF
firstLineOffset	Offset	int		0
fonts	Fonts	XeFont *	I	Default font as primary font
format	Format	Boolean		False
frameBottomRendition	FrameRendition	XeFrameBorder		1-pixel solid line
frameBottomContrast	FrameBottomContrast	int		40
frameInvertShadow	FrameInvertShadow	Boolean		False
frameLeftRendition	FrameRendition	XeFrameBorder		1-pixel solid line
frameRightRendition	FrameRendition	XeFrameBorder		1-pixel solid line
frameTopRendition	FrameRendition	XeFrameBorder		1-pixel solid line
frameTopContrast	FrameTopContrast	int		20
graphicRendition	Rendition	String	I	Default rendition (“0”)
indentation	Offset	int		0
initialState	Sequence	String	I	ISO 8859-1 (G0->LR, G1->GR)
itemization	Itemization	XeItemization *		No itemization
kerningOffset	KerningOffset	Dimension *	N	No kerning offsets
layoutCallback	LayoutCallback	Pointer		NULL
lineLayoutTable	LineLayoutTable	XeTabStop *		No tabulation stops for STAB
lineSpacing	LineSpacing	int		Computed from the default font
pageSeparation	PageSeparation	int		0
proportional	Proportional	Boolean		False

a. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial implementation in some widget classes. "I" in this column indicates resources in which changes will become effective only when the content is loaded/imported into the widget (that is, when **contentLoaded** resource is set to False).

Constraint resources:

Name	Class	Type	Default Value
bottomShadowContrast	BottomShadow Contrast	int	40
frameAlignment	FrameAlignment	XeFrameAlignment	XeFrameAlignment_ABOVE
invertShadow	InvertShadow	Boolean	False
horizontal	FrameDimension	XeFrameDimension	XeFrameDimension_MAX
leadingBorder	FrameBorder	XeFrameBorder	No border
leadingSeparation	FrameSeparation	int	int
leftBorder	FrameBorder	XeFrameBorder	No border
leftSeparation	FrameSeparation	int	0
framePosition	FramePosition	XeFramePosition	XeFramePositionType_INLINE
rightBorder	FrameBorder	XeFrameBorder	No border
rightSeparation	FrameSeparation	int	0
topShadowContrast	TopShadowContrast	int	20
trailingBorder	FrameBorder	XeFrameBorder	No border
trailingSeparation	FrameSeparation	int	0
vertical	FrameDimension	XeFrameDimension	XeFrameDimension_MAX

This widget supports a subset of the features from ISO 6429[3]. Among other things the controls can be used to select graphic renditions (italicized, underlined, etc.), multiple fonts and point sizes. The widget can be told to treat the text as *formatted* or *processable*. When treated as formatted, the widget will not break lines until it sees an [CR]LF combination, when treated as processable, long lines will be wrapped by the widget.

3.1 Defined resources

alignment Defines the alignment of the lines within widget. The following values are possible: `XeAlignment_START`, `XeAlignment_END`, `XeAlignment_CENTER` and `XeAlignment_JUSTIFIED`. These have the same semantics as the **alignment** attribute in ODA[4] standard. The XeAlignment enumeration defines these and some other values.

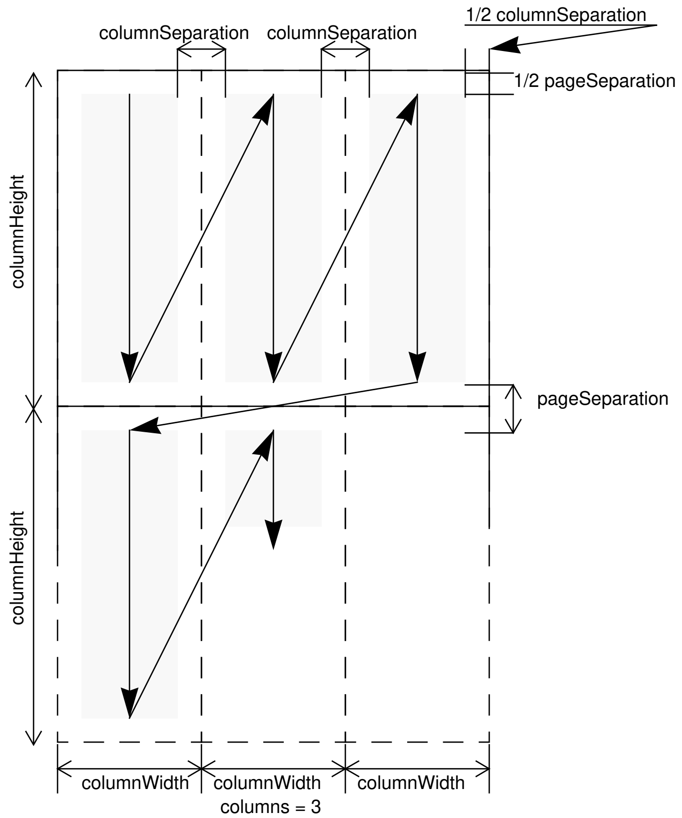
XeAlignment	
Enumeration value (name)	**Converted from String**
`XeAlignment_NONE`	none
`XeAlignment_START`	start
`XeAlignment_END`	end

XeAlignment	
Enumeration value (name)	**Converted from String**
`XeAlignment_CENTER`	center
`XeAlignment_JUSTIFIED`	justified
`XeAlignment_AROUND`	around

columnHeight

when non-zero, this specifies the column height (*in pixels*) to be used in laying out the content of the widget. If the value is zero and the computed **columns** is more than one, the current **height** of the widget window is used. If computed columns is one, the column height becomes infinite, if no explicit value is specified. The actual height to set the text is **pageSeparation** subtracted from the computed column height.

The following diagram shows the relationships between the resources: **columns**, **columnHeight**, **columnSeparation**, **columnHeight** and **pageSeparation**.



*Xew-4.0 has a bug: when the **columnHeight** is specified as zero, it will default incorrectly to the window height when **columns**=1 (the default).*

*The work around to disable paging in 4.0 is to use explicitly very large number for the **columnHeight**.*

columns when non-zero, this specifies the number of columns to set the text. For this to be effective, the column height must determinable.

columnSeparation
is evenly divided between the left and right free space *within* the text column (the amount of empty space to be left on the sides of the text).

columnWidth
when non-zero, this specifies the column width (*in pixels*) to be used in laying out the content of the widget. If the value is zero, the column width is computed from the current **width** of the widget window and current value of the **columns** resource. The actual *line width* to set the text is **columnSeparation** subtracted from the computed column width.

exportFormat
controls the type of octet stream that is generated when parts of the text content is exported out of the widget either by cut, selections or programmatic interface functions.

XeTextExport		
Enumeration value (name)	**Converted from String**	**Target Atom (with selections)**
`XeTextExport_STRING`	string	STRING
`XeTextExport_STRING_F`	string-f	STRING
`XeTextExport_ODIF`	odif	COMPOUND_TEXT
`XeTextExport_ODIF_F`	odif-f	COMPOUND_TEXT
`XeTextExport_ODIF_FP`	odif-fp	COMPOUND_TEXT

The exported character stream can either be *source text* or *formatted text* (.._F symbols). The source text will only include the original text content, the formatted text includes the line breaks and other possible information inserted by the layout process.

If the content or part of content is copied into CUT_BUFFER, or if the selection requests for a STRING target, the generated stream will be either `XeTextExport_STRING` or `XeTextExport_STRING_F`. If the selection requests COMPOUND_TEXT or TEXT target, the resource value will determine the format of the stream.

`XeTextExport_STRING, XeTextExport_STRING_F`

STRING (Latin 1) source or formatted text. The stream will contain only ISO 8859-1 characters and control characters HT (0x09) and LF (0x0a).

`XeTextExport_ODIF, XeTextExport_ODIF_F`

COMPOUND_TEXT source or formatted text. The stream will contain the text content and all ISO 2022 and ISO 6429 control and escape sequences that are necessary to describe the content (see section 3.7 on page 41 for the current supported list).

The generated stream contains more escape and control sequences than is described in the MIT Compound Text Encoding document [5].

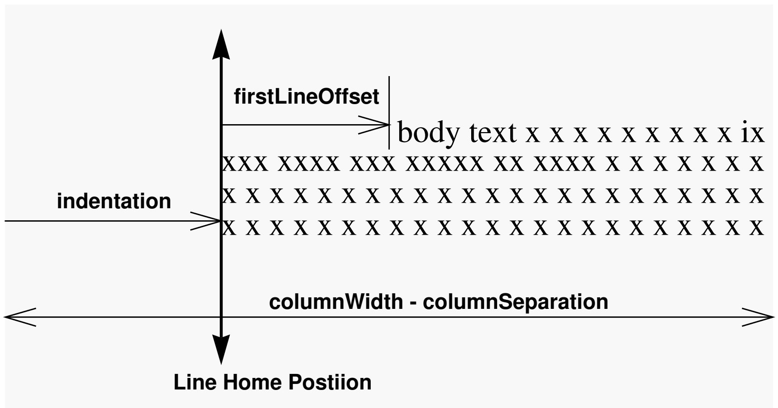
`XeTextExport_ODIF_FP`

ODIF formatted processable character content. The stream has the source content as in XeTextExport_ODIF, and in addition all the information generated by the layout process enclosed within SOS — ST delimiters.

The generated octet stream will contain all the control sequences that have been used in the content. If the application needs the output to be ODA/ODIF compatible, it must not use features not supported by the target ODA profile.

firstLineOffset

specifies the first line offset as defined in ODA standard, except that the offset is defined in pixels.



The **firstLineOffset** affects the layout of the first line in the paragraph and is defined to be the relative distance from the *line home position* (the value can be either positive or negative).

fonts describes the fonts that can be used in the text belonging into this widget. This resource value is a pointer to an array of **`XeFont`** structures. This array can contain at most 10 entries, the first of which is called *primary (default) font* and the rest are *alternative fonts* from *first* to *ninth*. These entries correspond the font selections that can be made by **Select Graphic Rendition** (SGR) control function. Each entry contains two fields: `font_size` and `font_list`. The latter is an internal identifier which is associated with a fontlist, that will determine correct font. The end of the array is indicated by the element having zero value in the `font_size` field.

```
typedef struct XeFont
    {
        int font_size;
        XeFontList font_list;
    } XeFont;
```

Currently, the easiest way to set this resource is through the resource converter. A simple example using only the primary font is

```
*XeText.fonts:16 -*-courier=iso8859-1,-
```

The Text class registers a `String` to `XeFont*` converter. This converter will create an array of XeFont structures, and associates a `font_list` with each specified X11 font specification.

<fonts>	::= <font> [";" <fonts>]
<font>	::= <size> [" " <fontlist>]
<fontlist>	::= <fontstring> ["=" <characterset>] ["," <fontlist>]
<fontstring>	::= <Any X11 font specification string>
<size>	::= <positive integer>
<characterset>	::= <Any supported X11 characterset name> | "default-1" | "default-2"

<size> specifies the desired font height in pixels. This size will be the default size of the text. Different sized variations from the same font can be achieved by using the **Graphic Size Modification** (**GSM**) control sequence.

<fontstring> is a string that is matched against all font names available on the X server. The string implicitly assumes a wildcard “*” at end (thus, a plain “-” will match all font names starting with “-”). If the match string does not begin with “+” or “-”, the string is assumed to select a non-XLFD font and it is used as such, without any further modifications.

`<characterset>`

is the X11 font encoding name:

Character Set Names	
Standard reference	**Character Set (X11 name)**
`ISO 8859-`*`n, (n = 1 .. 9)`*	iso8859-*n*
`JIS X0201-1976`	jisx0201.1976-0
`JIS X0208-1983, GL encoding`	jisx0208.1983-0
`GB 2312-80`	gb2312.1980-0
`KS C5601-1987, GL encoding`	ksc5601.1987-0

In addition to the above, two special names `default-1` and `default-2` are recognized. The ISO-2022 parser will put all unrecognized single byte character sets into `default-1` and unrecognized multibyte character sets into `default-2`.

For example, if the X server has the BIG5 font "taipeik20" installed, the following two resource definitions transform the text widgets into BIG5 widgets

```
*.XeText.fonts: 20 taipeik20=default-2,-
*.XeText.initialState: \033(B\033$*Z
```

A font list (**`XeFontList`**) is has two functions: first it informs the widget about the actual fonts to be used, and second it can be used to bind a character set into specific font.

XeFontList serves a similar function as XmFontList in MOTIF, but is not same. The MOTIF FontList operates with XFontStruct and is display specific. Xew font list operates with X11 font names in string form and is not yet bound to any specific display or font size.

For example, times may apply to European scripts, but there may not be any times font for the Japanese Kanji script. Thus a following resource definition for fonts could be used:

```
*.fonts: 16 -abode-times*=iso8859-1,-
```

The above would try to use times font only for characters coded according to ISO 8859-1 and use whatever font it can find that matches the japanese graphic characters (for example JIS X0208-1983).

The font string should have sufficient wild cards, especially it should not specify the size. The specific size of font will be searched among the fonts specified with font string, and if the size is included in that string, it makes the size part ineffective.

If an application needs to build the **fonts** array internally, it can convert the string representation of a `<fontlist>` into **XeFontList** using the Xew library function **XeMakeFontList**.

*The **fonts** array must exist as long as it is assigned to any widget. The current version (Xew 4.0) does not make a private copy of the array. The same array can be shared by multiple widgets.*

format is a Boolean flag which tells whether the content needs formatting before presentation. False (the default), indicates that the lines are kept as they are. When the value is True, the lines are folded to the *line width*, which is determined from the resource **columnWidth** directly, or computed based on the other page layout resources. If even the computed *line width* is zero, no folding of lines occur and the widget behaves like this resource had value False.

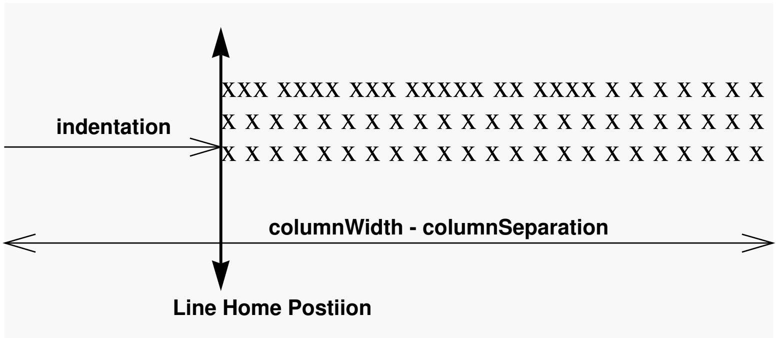
frameTopRendition, **frameBottomRendition (FrameRendition)**

frameLeftRendition, **frameRightRendition** (**FrameRendition**)

frameTopContrast, **frameBottomContrast, frameInvertShadow**
(**FrameRendition**, **FrameTopContrast**, **FrameBottomContrast**, **FrameInvertShadow**) are a group of resources giving a very detailed control over the visual rendition of the "framed rendition" (**SGR**). The default value produces a simple solid box around the text. (The possibilities of these resources are the same as is defined for **FrameBorder**, **TopShadowContrast**, **BottomShadowContrast** and **InvertShadow** in Frame.)

graphicRendition
is a character string forming the parameter values to SGR sequence, which define the state of the rendition attributes at the beginning of the text content, for example "1;4", to force locked **bold underlined** text for the widget content (See also, initialState below).

indentation is the amount of text indentation from the left edge of the layout area and determines the *line home position* within the text column.



initialState is an octet string containing ISO 2022 and ISO 6429 escape sequences which set up the initial state of G0-G3 and graphic rendition. If not defined, a state conforming with ISO 8859-1 is assumed. This default state can be described with following:

ISO Registration number	Character Set	Designated into	Invoked into
6	ISO 646, USA Version X3.4 -1968	G0	GL
6	ISO 646, USA Version X3.4-1968	G1	—
100	ISO 8859-1, Right-hand Part of Latin Alphabet No.1	G2	GR
100	ISO 8859-1, Right-hand Part of Latin Alphabet No.1	G3	—

The above state could also be achieved by defining the following sequence of designations and invocations into **initialState** resource:

```
ESC ( B ESC ) B ESC . A ESC / A LS0 LS2R
```

The content of initialState is not checked, at this point it can contain any control sequences and even normal graphic characters. Whether this will be supported or not in the final version is still open. The current semantics is the side effect of the implementation, which firsts interprets ***initialState*** *and then executes* ***graphicRendition****.*

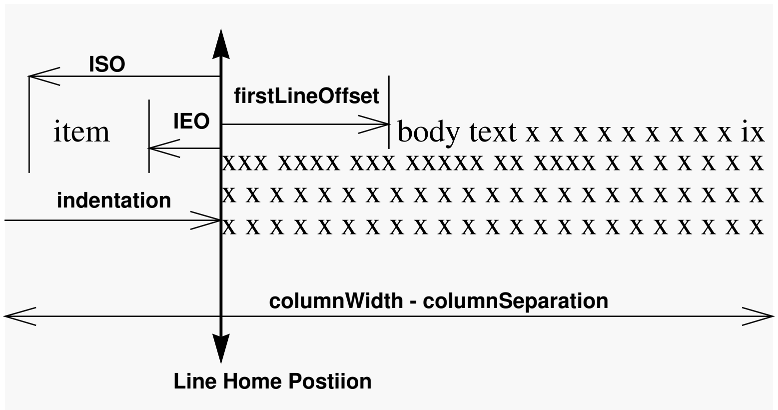
itemization specifies the itemization as defined in ODA standard. The resource value is a pointer to a structure

```
typedef struct XeItemization
    {
        int identifier_alignment;
        Dimension identifier_start_offset;
        Dimension identifier_end_offset;
    } XeItemization;
```

The widget makes a private copy of this structure.

Itemization is one of the resource values that effect the layout of the first line of a paragraph, other related resources are the **firstLineOffset** and **indentation** and **columnWidth**. The following diagram shows the combined effect of all these, when the itemization is in effect (`identifier_alignment != XeAlignment_NONE`).

ISO = identifier_start_offset (in this example, negative value shown)
IEO = identifier _end_offset (in this example, negative value shown)



When itemization is enabled, the string that will get treated as an *item identifier* is all text from the beginning of the paragraph content until the first TAB, or any other *sequence* ending character (for example, CR, LF, Quad codes, etc.). The *item identifier* string is independently aligned according to the `identifier_aligment` specification to the markers defined by the `identifier_start_offset` and `identifier_end_offset`.

`identifier_alignment == XeAlignment_NONE (or illegal value)`

itemization effect is disabled and only the resources **firstLineOffset**, **indentation** and **columnWidth** are in effect for the paragraph start.

`identifier_alignment == XeAlignment_START`

itemization is enabled and the start edge of the *item identifier* string is aligned to the position determined by the `identifier_start_offset`.

`identifier_alignment == XeAlignment_END`

itemization is enable and the end of edge of the item identifier string is aligned to the position determined by the `identifier_end_offset`.

`identifier_start_offset`

is a *signed integer* specifying the identifier start offset position *relative* to the *line home position*.

`identifier_end_offset`

is a *signed integer* specifying the identifier end offset position *relative* to the *line home position*.

The Text class registers a `String` to `XeItemization*` converter with the following syntax:

<itemization>	::= <alignment> ["," [<start>] ["," <end>]]
<alignment>	::= "none" | "start" | "end"
<start>	::= <integer>
<end>	::= <integer>

layoutCallback

is called by the layout process to obtain additional information from the application. The following callbacks are currently defined:

`XeCR_LAYOUT_CHANGE_TAG` and `XeCR_LAYOUT_BEGIN_LINE`

`XeCR_LAYOUT_CHANGE_TAG` is called when the layout process goes over a change of the tag associated with the source content. `XeCR_LAYOUT_BEGIN_LINE` is called at the beginning of every line, just before the source content is laid out. Both callbacks have the following information as callback_data

```
typedef struct XeTextLayoutCallbackData
    {
        int reason;             /* XeCR_LAYOUT_CHANGE_TAG */
                                /* XeCR_LAYOUT_BEGING_LINE */
        XeTextTag old,new;      /* Old and new Tag value */
        XeTextInsertContext context; /* Insert Context */
        int (*feed)(XeTextInsertContext, char *, long);
    } XeTextLayoutCallbackData;
```

By calling the supplied **`feed`** function the application can insert layout content into the widget at this point. All control sequences that are allowed for the normal content can be used here.

Tags are discussed in more detail in connection with TextEd widget class.

lineLayoutTable

is a pointer to an array of tabulation stop definitions. The end of the array is indicated by an element having a negative value in the `reference` field.

```
typedef XeTabStop
    {
        int reference;
        int position;
        int alignment;
        char alignment_string[4];
    } XeTabStop;
```

The widget does not make a private copy of this resource value. The pointer is used and it must remain valid until new value is set or the widget is destroyed.

The Text class registers a `String` to `XeTabStop*` converter with the following syntax:

<table>	::= <tabstop> [";" <table>]
<tabstop>	::= <position> ["#" <reference>] ["," <alignment> ["," <around>]]
<reference>	::= <String composed of one to four digits>
<position>	::= <non-negative integer>
<alignment>	::= "start" | "end" | "center" | "around"
<around>	::= <Max 3 characters to align around (usually "." or ",")>

`reference` is optional, and if omitted, defaults to the sequence number of the stop being defined (numbering starts from 1). The reference is used by the **Selective Tabulation** (**STAB**) sequence.

`position` specifies the horizontal position of the tabulation stop. When positive, the value is absolute horizontal coordinate position; when negative, the position is relative to the right margin of the layout column. If position is ZERO and alignment is END, the tabulation stop is defined to be at the right margin.

`alignment` is optional and defaults to START alignment. This specifies how the tabulated text is aligned relative to the tabulation stop position.

`around` is used only if alignment is specified as AROUND. If the tabulated text contains any of these characters, then the first of them is aligned to the tabulation stop position. Currently at most 3 characters can be specified.

If the **lineLayoutTable** is defined, it overrides the normal **Horizontal Tabulation** (**HT**) processing. In that case, the HT will advance *forward to the closest defined tabulation position*. If the current position is already beyond all defined positions, the normal evenly spaced HT processing is resumed.

lineSpacing if the **proportional** line spacing is not specified, this resource defines the distance between the baselines (reference lines) of two consecutive lines of text. When the **proportional** line spacing is specified, the **lineSpacing** is only used for determining the forward and backward extents of empty lines (the lineSpacing is split into 1/3 and 2/3 for forward and backward extents respectively).

If the value is left undefined (zero), the default value is computed from the default font of the widget assuming the ISO 8859-1 encoding.

pageSeparation
is evenly divided between the top and bottom free space *within* the text column (the amount of empty space to be left above and below the text column).

proportional controls the proportional line spacing. If the value is False, the distance between baselines of lines is set by the **lineSpacing** resource. If the value is True, the distance between lines depend on their content. The default value is False.

3.2 Defined constraint resources

As a general rule, the constraint resources are the same as those defined for Frame, but with some semantic differences. These resources apply to the child widgets (*insets*) of Text.

frameAlignment
defines the desired positioning of the child object (inset) vertically within the line box. Text uses the same enumeration type as Frame for this resource, but partially different values are in effect:

```
typedef enum
    {
        XeFrameAlignment_LEFT,    /* XeFrame */
        XeFrameAlignment_RIGHT,   /* XeFrame */
        XeFrameAlignment_CENTER,  /* XeFrame,XeText */
        XeFrameAlignment_TOP,     /* XeText */
        XeFrameAlignment_BOTTOM,  /* XeText */
        XeFrameAlignment_ABOVE,   /* XeText */
        XeFrameAlignment_AROUND,  /* XeText */
        XeFrameALignment_BELOW    /* XeText */
    } XeFrameAlignment;
```

The placement of an inset is based on extents of the inset, which include the possible space allocations for the borders.

`XeFrameAlignment_ABOVE`
: the bottom edge of the inset is aligned on the baseline.

`XeFrameAlignment_AROUND`
: the center of the inset is aligned on the baseline.

`XeFrameAlignment_BELOW`
: the top edge of the inset is aligned on the baseline.

`XeFrameAlignment_BOTTOM`
: the bottom edge of the inset is aligned on the bottom edge of the line box.

`XeFrameAlignment_CENTER`
: the center of the inset is aligned on the center of the line box.

`XeFrameAlignment_TOP`
: the top of the inset is aligned on the top edge of the line box.

framePosition

defines the desired positioning of the child objects horizontally.

```
typedef enum
    {
        XeFramePositionType_NORMAL,  /* XeFrame */
        XeFramePositionType_REVERSE, /* XeFrame */
        XeFramePositionType_FIXED,   /* XeFrame */
        XeFramePositionType_INLINE,  /* XeText */
        XeFramePositionType_START,   /* XeText */
        XeFramePositionType_END      /* XeText */
    } XeFramePositionType;

typedef struct XeFramePosition
    {
        XeFramePositionType type;
        Position x, y;
    } XeFramePosition;
```

`XeFramePositionType_INLINE`

the inset flows along with other text. It is treated like a single character glyph within line (x and y are ignored).

`XeFramePositionType_START`

the inset is floated to the left edge of the text column at current line box and all following text will flow around it (x and y are ignored).

`XeFramePositionType_END`

the inset if floated to the right edge of the text column at current line box and all following text will flow around it (x and y are ignored).

`XeFramePositionType_FIXED`

the inset is placed at (x,y) position. Depending on which side of the inset there is more room, the text will flow around the inset similar to START or END positioning (see above).

horizontal, **vertical (FrameDimension)**

are hints or instructions to the frame about the desired dimensions of the child widget. The resource value is a structure defined as:

```
typedef struct XeFrameDimension
    {
        int value;
        int min, max;
    } XeFrameDimension;
```

The `value` field is interpreted as follows by the XeText `value > 0` if greater than zero, this gives the size of the child (constraint). The child will not have any choice about the value.

`value == 0` the dimension is the maximum available. For horizontal dimension, this will be the remaining space on the current line, and for vertical dimension this will be height of the line box that is computed up to this point.

This semantics is somewhat experimental, do not rely on this extensively.

`value < 0` the preferred dimension of the child is used.

invertShadow see **invertShadow** in Frame.

leadingBorder, **trailingBorder**, **rightBorder**, **leftBorder (FrameBorder)**
see **FrameBorder** in Frame.

bottomShadowContrast, **topShadowContrast**
see the **shadow contrast** specifications in Frame.

leftSeparation, **rightSeparation**, **trailingSeparation, leadingSeparation**
(**FrameSeparation**) gives the minimum distance from the text content, the extra space around the inset.

3.3 Use of resources from the superclass(es)

The resources from the Basic class are mostly used as described in the Basic class documentation, except

scaling has no effect on Text widget.

3.4 Translations and actions

The following actions are defined for the Text class:

Miscellaneous

`highlight()`

highlight the text widget content. The highlight method is chosen from the **frameTopRendition** resource. If “shadowed frame” is in effect, the highlight is done by changing the background to the same color as is defined for this edge. Otherwise highlight is done by drawing a one pixel wide rectangle inside the widget borders. (In the future, own recourse for controlling the highlight might be defined).

`unhighlight()`

cancel the highlight state set by highlight().

3.5 Installed converters

The Text class installs the following resource converters:

from_type	to_type	Description
XtRString	XtRXeAlignment	Convert string to **XeAlignment** value
	XtRXeTextExport	Convert string to **XeTextExport** value
	XtRXeFonts	Convert string to **XeFont** array
	XtRXeItemization	Convert string to **XeItemization** structure
	XtRXeLineLayoutTable	Convert string to **XeLineLayoutTable** array
Also defined are the resource converters needed for constraints		
XtRString	XeRFrameAlignment	Convert string to **XeFrameAlignment** value
	XeRFramePosition	Convert string to **XeFramePosition** structure
	XeRFrameDimension	Convert string to **XeFrameDimension** structure
	XeRFrameBorder	Convert string to **XeFrameBorder** structure

The resource names starting with "XtRXe..." should probably be renamed into "XeR..." for consistency.

3.6 Programmatic interface

`XeTextLayout(Widget)`

(Re)run the layout process for all of the content and redisplay. This should be used by the application when the layout information generated by the layoutCallback functions is changed.

3.7 Supported control sequences within the text content

The supported control functions serve two different purposes: first the code extension techniques defined by ISO 2022[2] are fully implemented, and secondly a selected subset of the ISO 6429[3] control sequences are supported. Both 7-bit and 8-bit forms are supported equally.

ISO 6429 (ANSI X3.64) Support

The control sequences from ISO 6429 presented below can be used within the text content. This list reflects the current state of implementation and additional support for new sequences may be added when need arises. Each description is title with bold line giving the mnemonic, expanded name and the actual representation of the control sequence (on the right hand side). Some controls may have two alternative presentations, one for 8-bit and another for 7-bit environments. Under the heading, a direct quote from the ISO 6429 standard is given in italics. Any additional paragraphs in normal text will describe implementation decisions and additional restrictions not dictated by the standard.

BPH—Break Permitted Here **0x82 or ESC 0x42**

BPH is used to indicate a point where a line break may occur when text is formatted.[1]

CR—Carriage Return **0x0d**

CR causes the active position to be moved to the line home position of the same line.

This control code is mostly ignored, if it appears alone. The only application of lone CR is marking the end of text portion that will be the item (see **itemization** resource). In all other cases CR must be followed by LF or FF.

CSI—Control Sequence Introducer **0x9b or ESC 0x5b**

CSI is used as the first character of a control sequence.

ESC—Escape **0x1b**

ESC is used for code extension purposes. It causes the meanings of a limited number of bit combinations following it in the data stream to be changed.

The use of ESC is defined in ISO 2022.

FF—Form Feed **0x0c**

FF causes the active position to be moved to the corresponding character position on a predetermined line of the next form or page.

The text content implements only the combined effect of CR+FF and a lone FF is not allowed (however, if this happens, the error recovery is that a preceding CR is implicitly assumed).[2]

GSM—Graphic Size Modification **CSI Ps Pn1; Pn2 0x20 0x42**

GSM is used to modify for subsequent text the height and/or the width of all primary and alternative fonts.

The text content implements only the height change (parameter Pn2 is ignored, if present), the width is defined by the selected height. The parameter value (pn1) specifies the height as percentage of the height selected by the **fonts** resource. The default value for Pn1 is 100.

1. The standard leaves open whether a hyphen should be added or not. This is because the decision is dependent on the language being used.

2. The current Widget Implementation treats FF equivalent to LF (semantics of "page" is not quite clear in this context).

HT—Character Tabulation **0x09**

HT causes the active position to be moved to the following character tabulation stop.

IS2—Information Separator Two (RS—Record Separator) **0x1e**

IS2 is used to separate and qualify data logically; its specific meaning has to be defined for each application. If this control is used in hierarchical order, it may delimit a data item called a record.

The text content uses IS2 (RS) as a paragraph separator. After RS the next line is treated like the first line of the content and the layout process reuses the resources **firstlineOffset** and **itemization**.

JFY—Justify **CSI Ps... 0x20 0x46**

JFY is used to indicate the beginning of a string of graphic character that are to be justified according to the layout specified by the parameter values.

In the text content JFY can be used to temporarily override the default text alignment of the widget (see the **alignment** resource). A section of text can be delimited by JFY sequences. The following table lists the state of the implementation:

Ps	Description
0 default	Ends the area of changed alignment and returns back to the default alignment of the widget as specified by the **alignment** resource.
1	*word fill*. Not implemented (currently ignored).
2	*word space*. Begin justified text section, the lines will be aligned by adding extra space into word spaces.
3	*letter space*. Not implemented (if used, acts like parameter 2).
4	*hyphenation*. Not implemented (ignored).
5	*flush to line home position margin*. Begin start aligned text section.
6	*centre between line home position and the line limit position margins*. Begin center aligned text section.
7	*flush to line limit position margin*. Begin end aligned text section.
8	*Italian hyphenation*. Not implemented (ignored).

If the sequence contains multiple conflicting parameters, the last one remains in effect.

Occurrence of JFY does not cause line break. The alignment specified by the JFY takes effect only if a line break occurs within the delimited section.

LF—Line Feed **0x0a**

LF causes the active position to be moved to the corresponding character position of the following line.

The text content implements only the combined effect of CR+LF and a lone LF is not allowed (however, if this happens, the error recovery is that a preceding CR is implicitly assumed).

NBH—No Break Here **0x83 or ESC 0x43**

NBH is used to indicate a point where a line break shall not occur when text is formatted. NBH may occur between two graphic character either or both of which may be SPACE.

Example A non-breaking SPACE can be represented with <NBH><SPACE> combination.

PLD—Partial Line Forward **0x8b or ESC 0x4b**

PLD causes the active position to be moved to the corresponding character position of an imaginary line with a partial offset in direction of the line progression. This offset should be sufficient either to image following characters as subscripts until the first following occurrence of PARTIAL LINE BACKWARD (PLU) in the data stream, or, if preceding character were images as superscripts, to restore imaging of following characters to the active line.

Example The sequence “m<PLD>2<PLU>” can be used to represent “m_2”.

PLU—Partial Line Backward **0x8c or ESC 0x4b**

PLU causes the active position to be moved to the corresponding characters position of an imaginary line with a partial offset in the direction opposite to that of the line progression. This offset should be sufficient either to image following characters as superscripts until the first following occurrence of PARTIAL LINE FORWARD (PLD) in the data stream, or, if preceding characters were imaged as subscripts, to restore imaging of following characters to the active line.

Example The sequence “m<PLU>2<PLD>” can be used to represent “m^2”.

QUAD—Quad **CSI Ps... 0x20 0x48**

QUAD is used to indicate the end of the a string of graphic characters that are to be positioned on a single line according to the layout specified by the parameter values.

In the text content QUAD can be used to align the preceding text sequence in a line. QUAD does not generate a new line and the same line can have multiple QUAD's (for example, to align part of the text to the start edge and part to the end edge).

Ps	Description
0 default	*flush to line home position margin*. Align the currently collected text to the start of the line.
1	*flush to line home position margin and fill with leader*. Not implemented (if used, works like parameter 0).
2	*centre between line home position and line limit position margins*. Align the currently collected text to the center of the line.
3	*center between line home position and line limit position margins and fill with leader*. Not implemented (if used, works like parameter 2).
4	*flush to line limit position margin*. Align the currently collected text to the end of the line.
5	*flush to line limit position margin and fill with leader*. Not implemented (if used, works like parameter 4).
6	*flush to both margins*. Justify the collected text in line.

Example The content "one two<QUAD0>three four<QUAD4>five six." might produce:

one two **three four**

five six.

SGR—Select Graphic Rendition **CSI Ps ... 0x6d**

SGR is used to establish one or more graphic rendition aspects for subsequent text. The established aspects remain in effect until the next occurrence of SGR in the data stream, depending on the setting of the GRAPHIC RENDITION COMBINATION MODE (GRCM).

The implementation assumes the GRCM = CUMULATIVE (that is, each aspect will be in effect until explicitly cancelled by another SGR).

Ps	Description
0	default rendition; cancel all preceding occurrences of SGR; invoke primary font.
1	bold or increased intensity
3	italicized
4	underlined
7	negative image
9	crossed out
10	primary (default) font
11-19	first to ninth alternative fonts (see also **fonts**)
21	doubly underlined
22	normal intensity (neither bold nor faint)
23	not italicized
24	not underlined (neither singly or doubly)
27	positive image
29	not crossed out
30-37	foreground color of the text; 30=black, 31=red, 32=green, 33=yellow, 34=blue, 35=magenta, 36=cyan, 37=white.
39	default foreground text color (foreground color of the widget)
40-47	background color of the text; 40=black, 41=red, 42=green, 43=yellow, 44=blue, 45=magenta, 46=cyan, 47=white.
49	default background text color (background color of the widget)
51	framed (see **FrameRendition**)
53	overlined [not implemented yet]
54	not framed, not encircled
55	not overlined [not implemented yet]

Example

The effect of making portion of text in “bold-italic” and can be achieved with the sequence “<SGR>1;3mbold-italic<SGR>22;23m”. Note that “;” acts as a parameter separator in the sequence.

STAB—Selective Tabulation **CSI Ps 0x20 0x5e**

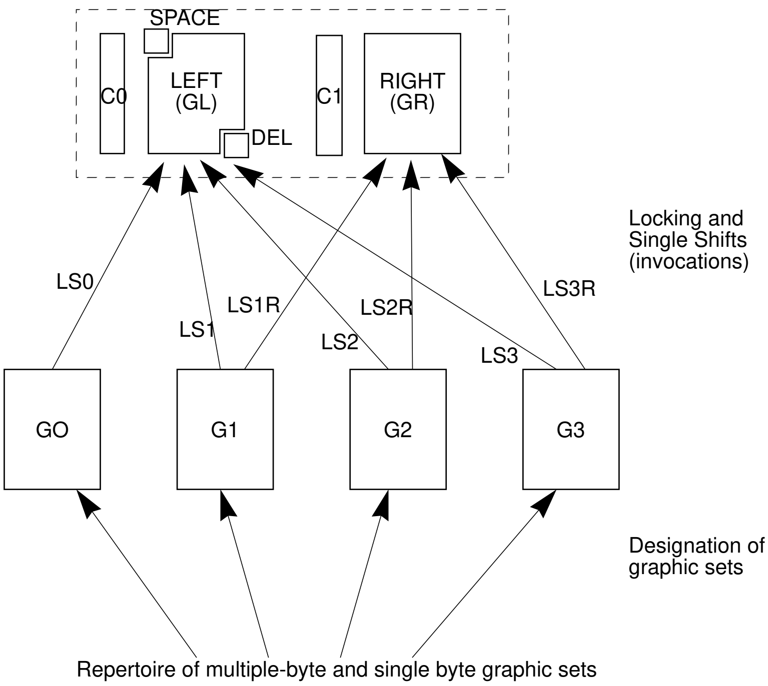
STAB causes subsequent text to be aligned according to the position and the properties of a tabulation stop which is selected from a list according to the value of the parameter Ps.

The tabulation positions are defined through **lineLayoutTable** resource. The semantics of this are taken directly from ISO 8613-6[4].

ISO 2022 Code Extension support

The text content implementation of code extension follows the 8-bit environment specified in ISO 2022. The 7-bit environment is a subset of this and is supported implicitly. The 8-bit code space is divided into 4 areas: **C0** (0x00-0x1f) and **C1** (0x80-0x9f) for control characters, **GL** (codes 0x21-0x7e) and **GR** (codes 0xa0-0xff) for graphic characters. There is no support for changing the default contents of C0 and C1 areas. Everything designated into G0, G1, G2 and G3 can be invoked into GL or GR (as specified in ISO 2022). The real support of the character sets that can be designated into G0-G3 to be used depend on implementation and available fonts on the X server.

The most useful place for the required invocation and designation sequences is the resource **initialState**. Normally the actual text content does not need to contain these[3].



3. There is at least one important exception to this: the specification for Japanese Character Encoding for Internet Messages is based on direct usage of designation sequences within the text content.

The designation sequences are given in the table below. "ESC x F", where F represents the *final character* selecting the character set.

Designation sequences				
Target	**94-char single byte**	**96-char single byte**	**94-char multibyte**	**96-char multibyte**
G0	ESC (F	not allowed	ESC $ F ESC $ (F	not allowed
G1	ESC) F	ESC - F	ESC $) F	ESC $ - F
G2	ESC * F	ESC . F	ESC $ * F	ESC $. F
G3	ESC + F	ESC / F	ESC $ + F	ESC $ / F
Non-ISO2022 with return	ESC % / F			
Non-ISO2022 no return	ESC % F			

All locking shifts (invocations) for graphic characters defined by ISO 2022 are supported. The implementation makes no differentiation between 8-bit and 7-bit controls and accepts both forms everywhere.

Invocation sequences (locking and shifts)			
Shift Function	**7-bit**	**8-bit**	**Invokes characters and side**
SHIF-OUT (SO, "\016")	0x0e	—	G1 to Left (GL)
SHIFT-IN (SI, "\017")	0x0f	—	G0 to Left (GL)
LOCKING-SHIFT ZERO (LS0, "\017")	—	0x0f	G0 to Left (GL)
LOCKING-SHIFT ONE (LS1, "\016")	—	0x0e	G1 to Left (GL)
LOCKING-SHIFT ONE RIGHT (LS1R, "\033~")	—	ESC 0x7e	G1 to Right (GR)
LOCKING-SHIFT TWO (LS2, "\033n")	ESC 0x6e	ESC 0x6e	G2 to Left (GL)
LOCKING-SHIFT TWO RIGHT (LS2R, "\033}")	—	ESC 0x7d	G2 to Right (GR)
LOCKING-SHIFT THREE (LS3, "\033o")	ESC 0x6f	ESC 0x6f	G3 to Left (GL)
LOCKING-SHIFT THREE RIGHT (LS3R, "\033 |")	—	ESC 0x7c	G3 to Right (GR)
SINGLE-SHIFT TWO (SS2, "\033N")	ESC 0x4e	0x8e	One Character from G2
SINGLE-SHIFT THREE (SS3, "\033O")	ESC 0x4f	0x8f	One Character from G3

The following tables describe the current implementation state of the designations:

ISO registration number	94-character graphic character sets with one intermediate byte	Final character	Mapping to X11 character set encoding (name)
6	ISO 646, USA Version X3.4 -1968	0x42 (B)	iso8859-1
10	ISO 646, Swedish Version, SEN 850200 B	0x47 (G)	iso8859-1 (conversion)
11	ISO 646, Swedish Version for Names, SEN 850200 C	0x48 (H)	iso8859-1 (conversion)
2, 4, 8-1, 8-2, 9-1, 9-2, etc...	All other registered ISO 646 national variants are currently mapped by default also into ISO 8859-1. Some simple conversion for the variant characters may later be implemented.	—	iso8859-1
14	ISO 646, Japanese Version for Roman Characters JIS C6220-1969	0x4a (J)	jisx0201.1976-0
13	Katakana Character Set JIS C6220-1969	0x49 (I)	jisx0201.1976-0
Others	For all unsupported 94 character sets, ISO 8859-1 left hand side is used as an ultimate fallback.	—	iso8859-1

.

ISO registration number	96-character graphic character sets with one intermediate byte	Final character	Mapping to X11 character set encoding (name)
100	Right-hand part of Latin Alphabet No.1, ISO 8859-1, ECMA-94	0x41 (A)	iso8859-1
101	Right-hand part of Latin Alphabet No.2, ISO 8859-2, ECMA-94	0x42 (B)	iso8859-2
109	Right-hand part of Latin Alphabet No.3, ISO 8859-3, ECMA-94	0x43 (C)	iso8859-3
110	Right-hand part of Latin Alphabet No.4, ISO 8859-4, ECMA-94	0x44 (D)	iso8859-4
144	Cyrillic part of the Latin/Cyrillic Alphabet?	0x4c (L)	iso8859-5
127	Right-hand part of Latin/Arabic Alphabet ECMA-114, ISO 8859-6	0x47 (G)	iso8859-6
126	Right-hand part of Latin/Greek Alphabet, Standard ELOT 928, ECMA-118, ISO 8859-7	0x46 (F)	iso8859-7

ISO registration number	96-character graphic character sets with one intermediate byte	Final character	Mapping to X11 character set encoding (name)
138	Latin/Hebrew Alphabet, ECMA-121	0x48 (H)	iso8859-8
148	Right-hand part of Latin Alphabet No.5	0x4d (M)	iso8859-9
156	Supplementary Set of ISO/IEC 6937	0x52 (R)	iso8859-1 (conversion)
Others	For all unsupported 96 character sets, ISO 8859-1 left hand side is used as an ultimate fallback.	—	iso8859-1

ISO registration number	94-character multiple-byte graphic character sets	Final character	Mapping to X11 character set encoding (name)
58	Chinese Character Set, Chinese Standard GB 2312-80	0x41 (A)	gb2312.1980-0
87	Japanese Character Set JIS C 6226-1983	0x42 (B)	jisx0208.1983-0
42	Japanese Character Set JIS C 6226-1978	0x40 (@)	jisx0208.1983-0 (fallback)
149	Korean Graphic Character, Korean Standard KSC 5601-1987	0x43 (C)	ksc5601.1987-0
Others	Designation ignored, except 2 byte codes assumed and whatever character set was in effect will be used	—	—

ISO registration number	Non ISO 2022 with standard return	Final character	Mapping to X11 character set encoding (name)
—	Currently no implementations	—	—

ISO registration number	Non ISO 2022 without standard return	Final character	Mapping to X11 character set encoding (name)
—	Currently no implementations	—	—

4. TextEd Widget

Application header file	<X11/Xew/TextEd.h>
Class header file	<X11/Xew/TextEdP.h>
Class	xeTextEdWidgetClass
Class Name	XeTextEd
SuperClass	XeText

A TextEd Widget is a subclass of Text and provides editing capabilities to all features supported by the Text widget. This class must also be used, if the selection support for the text content is required.

Name	Class	Type	Notes[a]	Default Value
allowEdit	AllowEdit	Boolean		True
cursorPosition	CursorPosition	long		0
displayCaret	Output	Boolean		True
modifyCallback	ModifyCallback	Pointer		NULL

a. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial implementation in some widget classes. "I" in this column indicates resources in which changes will become effective only when the content is loaded/imported into the widget (that is, when **contentLoaded** resource is set to False).

4.1 Defined resources

allowEdit allows editing of the content from the keyboard and mouse, when True. If the value is False, only editing possible is through the use of the programmatic interface.

cursorPosition
contains the cursor position expressed as virtual offset from the beginning of the content. This resource can be set and read.

displayCaret
indicates whether the widget is displaying the caret symbol at insert point (True) or not (False). The value of this resource can also be dynamically changed by `display-caret()` action.

modifyCallback (`XeCR_MODIFY_TEXT`)
is called whenever the content is about to be changed. The callback function has the option of cancelling the intended modification by setting the 'cancel' field in the callback data to True.

```
typedef struct XeTextModifyCallbackData
    {
        int reason;             /* XeCR_MODIFY_TEXT */
        Boolean cancel;         /* Initialized to False */
        XEvent *event;          /* Event or NULL */
    } XeTextModifyCallbackData;
```

4.2 Use of resources from the superclass(es)

All of the Text resources are usable.

4.3 Translations and actions

The interface descriptions use the following concepts:

sequence The text content is internally represented by logical sequences. The most simple sequence is a text that ends with a hard line-end, but a sequence does not need to end the line. A sequence is a unit of text that is aligned and placed “independently” on the line. Quad and horizontal tabulation end sequences that don’t change the line.

insert point The widget may have one open *insert point* or none. The *insert point* can be opened by **`XeTextReplaceTagged`** or by user starting some editing action. Once open, the *insert point* stays open until explicitly moved by **`XeTextReplaceTagged`** (a new insert point will be opened) or until the user does any cursor movement (insert point will just be closed).

When *insert point* is opened, the resource **initialState** defines the initial state of subsequent input. In addition, any explicit rendition and presentation settings in the **initialState** are locked and cannot be changed during input.

virtual offset Moving around within the content is based on *virtual offset* which defines the position as a distance from the beginning of the content as virtual character positions. The beginning of the content is represented by virtual offset zero. Each character (either single or multibyte) occupies one virtual character position. In addition, each end of sequence (line, tabulation mark, quads etc.) in source content count as one position (the content generated by the layout process has no effect on virtual offset).

The supported translations and actions follow very closely the MIT Athena Text widget conventions. The following table lists the actions from X11 release 5 Text widget and indicates which of them are implemented by this widget (this describes the current state of implementation).

Cursor Movement Actions

`forward-character(), backward-character()`

Move the cursor point forward or backward within content one virtual character position.

`forward-word(), backward-word()`

Move the cursor point to next or previous word boundary.

`forward-paragraph(), backward-paragraph()`

Paragraph moving not supported.

`beginning-of-line(), end-of-line()`

Move the cursor point to the beginning or end of the current line. If the point is already at end or beginning of line, no movement happens.

`next-line(), previous-line()`

Move the cursor point up or down one line. The new position will be the closest possible position to the horizontal starting point of consecutive next-line/previous-line actions.

`next-page(), previous-page()`

Page moving not supported.

`beginning-of-file(), end-of-file()`

Move the cursor point to the beginning or end of the content.

`scroll-one-line-up(), scroll-one-line-down()`

Scroll operations not supported.

New Line Actions

`newline-and-indent()`

Not implemented.

`newline-and-backup()`

Insert a hard line ending after the insert point.

`newline()`

Insert a hard line ending to the insert point (before insert point).

Delete Actions

`delete-next-character(), delete-previous-character()`

Remove the (virtual) character immediately after or before the insert point. Removing line or sequence end will join lines or sequences together.

`delete-next-word(), delete-previous-word()`

Remove characters between the insert point and next or previous word boundary. A word boundary is defined as a space or end of sequence.

`delete-selection()`

Remove all characters in the current selection.

Kill Actions

`kill-word(), kill-backward-word()`

Same as `delete-next-word` and `delete-previous-word`, but they store the deleted characters into the kill buffer (CUT_BUFFER1).

`kill-selection()`

Same as `delete-selection`, but store the deleted characters into the kill buffer (CUT_BUFFER1).

`kill-to-end-of-line()`

Delete the line from insert point up to (but not including) the end of line (either hard or generated). A special case handling occurs if the insert point is already at end-of-line, in that case deletes the end-of-line. The deleted characters are stored into the kill buffer (CUT_BUFFER1).

`kill-paragraph()`

Not implemented.

`kill-to-end-of-paragraph()`

Not implemented.

Selections Actions

`select-word()`

Not implemented.

`select-all()`

Select all of the content (place the selection start point to the beginning of the content, and end-point to the end of the content).

`select-start()`

Place the selection start point to the location of the event. This action will not change the location of the insert point (contrary to the Athena).

`select-adjust()`

Change the selection end point to the location of the event.

`select-end([name[,name, ...]])`

End the selection sequence, assert selection ownership and copy the selection to the cut buffer(s) as instructed by the parameter. Each `name` can be a selection (PRIMARY, SECONDARY, ...) or a cut buffer (CUT_BUFFER0, CUT_BUFFER1, ...). If no `names` are specified, PRIMARY is asserted.

`extend-start()`

Find the nearest end of the current selection and move it to the location of the event.

`extend-adjust()`

Same as `select-adjust`.

`extend-end([name[,name, ...]])`

Same as `select-end`.

`insert-selection(name[, name, ...])`

Retrieve the value of the first named selection that exists or the cut buffer that is not empty and insert this value into the current insert point. A `name` can be a selection (PRIMARY, SECONDARY, etc.) or a cut buffer (CUT_BUFFER0, CUT_BUFFER1, ...).

Miscellaneous Actions

`redraw-display()`

Redo the layout process for the content and redisplay the content. This is useful if for some reason the display and laid out content get out of synch.

`insert-file([filename])`

Not implemented.

`insert-char()`

This action may be attached only to a key event. It calls `XLookupString()` to translate the event into a character sequence and inserts that sequence into the text at the insert point. The actual interpretation of the character sequence depends on the current state of the insert point. For example, if the insert point is in Shift-JIS mode, then every two octets will cause single kanji character to be inserted.

`insert-string(string[, string, ...])`

Insert each `string` into the insert point. Any `string` beginning with the characters "0x" and containing only valid hexadecimal digits in the remainder is interpreted as a hexadecimal constant and the corresponding single character is inserted instead.

`display-caret([state[,when]])`

Controls the displaying of the “caret” symbol at insert point. The `state` (converted from string to Boolean) specifies the visibility of the caret: True makes caret visible and False hides it. The default value is True, if no arguments are given.
The `when` argument applies only, if this action is bound to `EnterNotify` or `LeaveNotify` events. If `when` is not present or is something other than “always”, the action is taken only if the focus field of the event is True. Possibly useful augmented binding might be:

```
*XeText.Translations: #override \
   <FocusIn>:  display-caret(on) \n
   <FocusOut>: display-caret(off)
```

`focus-in(), focus-out()`

Currently, these don’t do anything.

`search(direction[,string])`

Not implemented.

`multiply(value)`

Set multiplier for the next action. Many text actions repeat themselves as many time as the current multiplier value indicates. The default value is one and the value is reset to one after being used. The action sequence "multiply(5) delete-line()" will delete 5 lines. Multiply actions can be concatenated, the sequence "multiply(5) multiply(5)" will set the value to 25. If the value is a string “reset’, the multiplier value is reset back to one and a beep is sent to the display.

`form-paragraph()`

Not implemented. (Not applicable to this widget, because it keeps the content “formatted” all time).

`transpose-characters()`

Not implemented.

`no-op([action])`

Without parameter does nothing and can be used in overriding translations. With parameter string `RingBell` a beep is sent to the display.

Actions specific to the Xew TextEd widget `notify([param, ...])`

This action will call notifyCallback and passing the param strings as call data (see **XeNotifyCallbackData** for detailed information).

`pointer()`

Move the cursor point the position under pointer.

The default event bindings for the XeTextEd widget are:

```
static char translations[] =
"\
Ctrl<Key>A:            beginning-of-line() \n\
Ctrl<Key>B:            backward-character() \n\
Ctrl<Key>D:            delete-next-character() \n\
Ctrl<Key>E:            end-of-line() \n\
Ctrl<Key>F:            forward-character() \n\
Ctrl<Key>G:            multiply(Reset) \n\
Ctrl<Key>H:            delete-previous-character() \n\
Ctrl<Key>K:            kill-to-end-of-line() \n\
Ctrl<Key>L:            redraw-display() \n\
Ctrl<Key>M:            newline() \n\
Ctrl<Key>N:            next-line() \n\
Ctrl<Key>O:            newline-and-backup() \n\
Ctrl<Key>P:            previous-line() \n\
Ctrl<Key>T:            transpose-characters() \n\
Ctrl<Key>U:            multiply(4) \n\
Ctrl<Key>W:            kill-selection() \n\
Ctrl<Key>Y:            insert-selection(CUT_BUFFER1) \n\
Meta<Key>B:            backward-word() \n\
Meta<Key>F:            forward-word() \n\
Meta<Key>K:            kill-to-end-of-paragraph() \n\
Meta<Key>Y:            insert-selection(PRIMARY,
CUT_BUFFER0) \n\
:Meta<Key>d:           delete-next-word() \n\
:Meta<Key>D:           kill-word() \n\
:Meta<Key>h:           delete-previous-word() \n\
:Meta<Key>H:           backward-kill-word() \n\
:Meta<Key>\\<:         beginning-of-file() \n\
:Meta<Key>\\>:         end-of-file() \n\
:Meta<Key>]:           forward-paragraph() \n\
:Meta<Key>[:           backward-paragraph() \n\
~Shift Meta<Key>Delete: delete-previous-word() \n\
 Shift Meta<Key>Delete: backward-kill-word() \n\
~Shift Meta<Key>BackSpace: delete-previous-word() \n\
 Shift Meta<Key>BackSpace: backward-kill-word() \n\
<Key>Right:            forward-character() \n\
<Key>Left:             backward-character() \n\
<Key>Down:             next-line() \n\
<Key>Up:               previous-line() \n\
<Key>Delete:           delete-previous-character() \n\
<Key>BackSpace:        delete-previous-character() \n\
```

```
<Key>Return:           newline() \n\
<Key>:                 insert-char() \n\
<FocusIn>:             focus-in() \n\
<FocusOut>:            focus-out() \n\
<Btn1Down>:            select-start() \n\
<Btn1Motion>:          extend-adjust() \n\
<Btn1Up>:              extend-end(PRIMARY, CUT_BUFFER0) \n\
<Btn2Down>:            insert-selection(PRIMARY,
CUT_BUFFER0) \n\
<Btn3Down>:            extend-start() \n\
<Btn3Motion>:          extend-adjust()\n\
<Btn3Up>:              extend-end(PRIMARY, CUT_BUFFER0) ";
```

4.4 Programmatic interface

`XeTextChangeMode(Widget w, char *mode, long length, long amount)`

The function will add and lock the specified modes (`mode`, `length`) into insert point, and then change the `amount` number of characters to match the locked modes. Only those modes that are locked will be changed, not all modes that are currently set at insert point.

The function does not move insert point, unless the `mode` string contains graphic characters (the mode string is processed exactly as with **XeTextInsert**, but with the side effect that any mode changing controls will also modify the locked attributes and is merged into the set of locked attributes (which already will contain attributes defined from **initialState** resource or from current **XeTextInsertPrefix**).

This function can be used in different ways. A simple use would be

```
XeTextChangeMode(w, "\033[31m", 5L, 20L);
```

which will change the foreground rendition to RED for the next 20 virtual positions counting from the current insert point forward.

`XeTextChangeModeAndTag(Widget w, char *mode, long length, long amount, XeTextTag tag)`

is similar to **XeTextChangeMode**, but changes also the tag value on the specified range

`void XeTextEnableDisplay(Widget w)`

`void XeTextDisableDisplay(Widget w)`

Disable and enable the layout and display update processing. When an application is doing multiple update actions into the text content, it is useful to disable updates to the display during the operations. Such actions can be enclosed within a pair of **XeTextDisableDisplay** and

XeTextEnableDisplay calls. The functions maintain a counter for the depth, and display is only enabled after the outermost pair is exited.

`void XeTextDisplay(Widget w)`

Force the redisplaying of current content, regardless of the Enable/Disable display status.

```
void XeTextExtract(Widget w, long start, long end,
        XeTextExtractFeed feed, XtPointer data)
```

Extract a range of content using the supplied function (**feed**). The format of the octet stream is determined from the **exportFormat** resource. This interface is the only way to extract the content with application specified tag information. The octet stream is released to the application by repeated calls to the feed function, which must have the following definition:

```
typedef int (*XeTextExtractFeed)(char *text,
        int length,XeTextTag tag, XtPointer data);
```

Each call to the **feed** function gives a snippet of text content (**text**, **length**) and the **tag** value associated with it. The **data** is the user supplied **data** from **`XeTextExtract`**. The division to snippets is completely dependent on the internal representation of the content and the application should not give any significance to snippet divisions. Zero tag is used for portions which no tag has been supplied. The **feed** function should return the actual number of bytes processed, returning less than **length** indicates an error which terminates further calls to the **feed** function and causes immediate return from the **`XeTextExtract`**.

The **feed** function must not modify the source **text**.

See also **`XeTextGetString`** for simpler access.

`long XeTextGetInsertionPosition(Widget w)`

Return the current insert position (**cursorPosition**).

*This is old **XeTextGetInsertionPoint** renamed to align with the other related function names.*

`XeTextTag XeTextGetInsertionTag(Widget w)`

Return the current value of the tag at *insert point* (or if insert point is not open, return the tag effective under the cursor).

`void XeTextGetRegionPosition(Widget w,int id,long *start,long *end)`

Return the text range (**start**, **end**) of the specified text region (**id**). If the returned **start** and **end** are equal, then the region does not exist. The returned **start** will always be less or equal to the returned **end**. See also **`XeTextSetRegion`** and **`XeTextUnsetRegion`**.

`String XeTextGetString(Widget w, long *length)`

Save the current content of the Text widget (**w**) in to a String. The format of the exported octet stream is determined by the **exportFormat** resource. The function returns a pointer to String and the actual length of the content in bytes is returned into the **length** parameter. Returns NULL, if the content is empty. The returned String must be released with **`XtFree`**, when the application does not need it any more.

For easy use in simple applications, the actual memory allocation for returned String is **length**+1 and this extra byte is set to zero.

For a more complex interface, see also **`XeTextExtract`**.

`void XeTextGetSelectionPosition(Widget w, long *start, long *end)`

Return the currently selected text range (**start**, **end**). If the returned **start** and **end** are equal, no selection is currently active. The returned **start** will always be less or equal to the returned **end**.

`String XeTextGetSubstring(Widget w, long *length, long start, long end)`

Same as **`XeTextGetString`**, except only the content between the specified virtual positions (**start**, **end**) are returned.

`void XeTextInsert(Widget w, char *text, long length)`

Insert new content (**text**, **length**) into the current *insert point* (if no *insert point* is open, one is opened at the current **cursorPosition**). This function can be used to add more input to the *insert point* opened earlier, it does not cause the input state being reset from the **initialState** like **`XeTextReplaceTagged`** does. See also **`XeTextSetInsertionTag`**.

`void XeTextInsertPrefix(Widget w, char *value, long length)`

Specify a control string (**value**, **length**) that will be executed every time after an insert point is opened, before any other data is inserted. The graphic rendition modes set by this sequence will be locked in the same way as **initialState** modes (insertPrefix can be used to override initialState).

`long XeTextLocateTag(Widget w, long position, XeTextTag tag, int mode)`

starting from the specified `position`, search for the specified `tag` (`mode == 0`) or different tag (`mode == 1`). Returns the virtual position if search succeeds (>= 0), or -1 if it fails.

`void XeTextReplace(Widget w, long start, long end, char *text, long length)`

Replace a range of content with new content. This function is just short for

```
XeTextReplaceTagged
        (w,start,end,text,length,(XeTextTag)0)
```

`void XeTextReplaceTagged(Widget w, long start, long end, char *text, long length, XeTextTag tag)`

Replace a range of content (**start**, **end**) with new content (**text**, **length**) and open a new *insert point*. The function first closes previous *insert point* (if open) and starts a new at the specified location. The new insert point starts in the state defined by **initialState** (and **graphicRendition**) resource. The opened insert points stays open until next **`XeTextReplaceTagged`** or until user causes the cursor to move from keyboard or from mouse.

*Application cannot insert control sequences with two separate **XeTextReplaceTagged** calls and expect their effect to be cumulative. Each **XeTextReplaceTagged** restarts the insert state from the **initialState**. To achieve cumulative effect, open the inserting with **XeTextReplaceTagged**, and continue pouring content to the same insert point with **XeTextInsert** and **XeTextSetInsertionTag** functions.*

`Boolean XeTextSaveAsFile(Widget w, String name)`

Save the current content of the Text widget (**w**) into a file (**name**). The format of the exported octet stream is determined from the **exportFormat** resource. The function returns True, if save succeeds and False otherwise.

`long XeTextScan(Widget w, XeTextUnit unit, int amount)`

scans forward or backward an **amount** of text units starting from the current position and returns the virtual offset value. This function does not change the current position. The unit is one of the values from the enumeration:

```
typedef enum
    {
        XeTextUnit_POINT,     /* Current Point */
        XeTextUnit_POSITION,  /* Virtual Position */
        XeTextUnit_WORD,      /* Word */
        XeTextUnit_LINE,      /* Line */
        XeTextUnit_LINE_BEGIN,/* Begin of line */
        XeTextUnit_LINE_END,  /* End of line */
        XeTextUnit_PARAGRAPH, /* Paragraph */
        XeTextUnit_TAG,       /* Same TAG */
        XeTextUnit_ALL        /* All Content */
    } XeTextUnit;
```

`void XeTextSetInsertionTag(Widget w,XeTextTag tag)`

Change the current **Tag** at insertion point (an insertion state is opened with defaults at current point, if none is previously open). This function can be used to change the Tag on insertion point without causing a reset of the input state, which would happen if one tried to change the tag with **`XeTextReplaceTagged`** function. See also **`XeTextInsert`**.

`int XeTextSetRegion(Widgetw,long start,long end,XeTextHighlight mode)`

Create a new application text region and return the allocated text region identifier. Returns ZERO, if creating the region fails. The widget maintains region **start** and **end** positions during the editing process. See also **`XeTextGetRegionPosition`** and **`XeTextUnsetRegion`**.

The **mode** parameter is defined by enumeration:

```
typedef enum
    {
        XeTextHighlight_UNUSED,
        XeTextHighlight_NONE,
        XeTextHighlight_REVERSE,
        XeTextHighlight_FRAME
    } XeTextHighlight;
```

`XeTextHighlight_NONE`

no visual effects are associated with the region.

`XeTextHighlight_REVERSE`

the region is highlighted with reverse video.

`XeTextHighlight_FRAME`

the region is highlighted by drawing a one pixel width rectangular polygon.

`void XeTextSetSelection(Widget w,long start,long end)`

Set/change the current selection to a new range (**start**, **end**).

`int XeTextUnsetRegion(Widget w, int id)`

Unset the text region (**id**) and return the highest identifier of the remaining allocated text regions, or return ZERO, if none left. See also **`XeTextSetRegion`** and **`XeTextGetRegionPosition`**.

`void XeTextUnsetSelection(Widget w)`

Clear the current selection.

5. Raster Widget

Application header file	<X11/Xew/Raster.h>
Class header file	<X11/Xew/RasterP.h>
Class	xeRasterWidgetClass
Class Name	XeRaster
SuperClass	XeBasic

Raster Widget is a rectangular area, that holds an image presented in one of the supported raster image interchange formats. The image can be either scaled or not scaled into the area. If image is larger than the area, it can be cropped by the limits of the area, or if scaling is requested it can be fitted into the area, either with or without preserving the aspect ratio (see the Basic class resources **scaling** and **resize**).

Name	Class	Type	Notes[a]	Default Value
clipX	ClipX	long		0
clipY	ClipY	long		0
clipWidth	ClipWidth	long		rasterWidth
clipHeight	ClipHeight	long		rasterHeight
rasterWidth	RasterWidth	long		Width from interchange format
rasterHeight	RasterHeigth	long		Height from interchange format

a. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial implementation in some widget classes. "I" in this column indicates resources in which changes will become effective only when the content is loaded/imported into the widget (that is, when **contentLoaded** resource is set to False).

5.1 Defined resources

clipX, clipY, clipWidth, clipHeight

These determine the rectangular area of pixels from the image that are to be displayed as a content of the widget. The original image is seen as a coordinate system where the top leftmost pixel has address (0,0) and bottom rightmost pixel has the address (*number of pixels in row* - 1, *number of rows in image* -1). The rectangle selected is defined by points (**clipX, clipY**) and (**clipX+ clipWidth**-1, **clipY+clipHeight**-1) in this coordinate space.

offsetX, offsetY

These resources have been deleted since 4.0 release. The equivalent function can be achieved by use of the Basic **xTranslation** and **yTranslation** resources.

rasterWidth, rasterHeight

specify the actual number of columns and rows of pixels in the raster image. These are read-only resources and the values are determined from the interchange format.

5.2 Use of resources from the superclass(es)

Many of the imaging related resources of the Basic widget are used. They are located in the Basic only because these resources are shared between Video and Raster widgets.

colormapInstall, colormapUse, colorMode, colorQuantize, maxColors
control the colour handing of the images and are supported as described in the Basic documentation.

mirrorImage, rotation, scaling
control the geometry of the image and are supported as described in the Basic documentation.

notifyCallback (`XeCR_RASTER_IMAGE`)
is called after an image has been imported into widget (converted into **`XeRawImage`**), but the display representation is not yet generated. The callback data structure is:

```
typedef struct XeRasterImageCallbackData
    {
        int reason;             /* XeCR_RASTER_IMAGE */
        int width, height;      /* Image Dimensions */
    } XeRasterImageCallbackData;
```

This callback gives the application a chance to modify the imported image, or if no image is imported (`width == 0 && height == 0`), the application can supply it at this point.

An application can provide own image import functions by specifying the **contentFormat** as `XeContentFormat_UNKNOWN` and performing the private import conversion in this callback. The callback must create the `XeRawImage` structure and attach it to the widget (see **`XeCreateRawImage`** and related functions).

useShm, visual
are also supported as described in the Basic documentation.

5.3 Translations and actions

Currently the Raster widget inherits the translations and actions from the Basic widget class.

5.4 Programmatic interface

raw image structure (XeRawImage)

the content of the **XeRaster** widget is stored into structure, which accessible to the application through a set of functions described below. The same structure can be the content of several widgets at same time and a reference count is kept in the structure. The `references` field in the structure must not be referenced or changed by the application.

`XeRawImage *XeCreateRawImage(int channels)`

Create empty image structure for the specified number of channels. Channels is usually 3 for colour images and 1 for grayscale or bitmap images. The returned structure must be released using the **XeDestroyRawImage**. The reference count is set to one.

`XeRasterDisplay(Widget w)`

calls the expose method of the widget. This function can be used, for example after **XeRasterSetRawImage**, if an immediate update is wanted (without this call, the display is only updated at by expose events).

This function does not exist on standard Xew-4.0 version, but is available as an add-on patch. It will be present in any future versions.

`XeRawImage *XeRasterGetRawImage(Widget w)`

Extract the `XeRawImage` structure currently attached to the **XeRaster** or return NULL, if none. The reference count is incremented by one and an application must call **XeDestroyRawImage** once for each **XeRasterGetRawImage**.

`void XeRasterSetRawImage(Widget w, XeRawImage *raw)`

Set new `XeRawImage` as a content of the **XeRaster**. This will cause the **XeRaster** to recompute the display version from the given new raw image. If the widget content had different attached raw image, then this will be destroyed (by **XeDestroyRawImage**) and the new raw image is installed and reference count is incremented.

*Currently Xew does not do any automatic refreshes to all widgets that have this same raw image as a content. But, application can keep track of widgets using the same content and call **XeRasterSetRawImage** for each of them to cause refresh of the display.*

`void XeDestroyRawImage(XeRawImage *raw)`

Destroy raw image structure. The reference count is decremented and only if it reaches ZERO, the actual structure is destroyed.

5.5 TIFF format (XeContentFormat_TIFF)

The TIFF format is supported through libtiff program library, if the library is available. The selection is made at compile time. The most common TIFF variations are supported. The “tiled” TIFF variants are not.

5.6 GIF format (XeContentFormat_GIF)

The GIF format is always supported.

5.7 JPEG format (XeContentFormat_JPEG)

The JPEG format is supported though the IJG jpegsrc library, if the library is available. The selection is made at compile time.

5.8 PBM format (XeContentFormat_PBM)

All six variants of the PBM format are supported (Ascii/Raw PBM, Ascii/Raw PGM and Ascii/Raw PNM).

6. Geometric Widget

Application header file	<X11/Xew/Geometric.h>
Class header file	<X11/Xew/GeometricP.h>
Class	xeGeometricWidgetClass
Class Name	XeGeometric
SuperClass	XeBasic

Geometric Widget is a rectangular area that holds a representation of graphics in one of the supported geometric interchange formats. The graphics can be either scaled or not scaled into the area. If graphics is larger than the area, it can be cropped by the limits of the area, or if a scaling is requested, it can be fitted into the area, either with or without preserving the aspect ratio.

This widget will be defined in similar ways as Raster Image Widget. The detailed description will supplied later.

This widget has not been implemented.

7. Animation Widget

Application header file	<X11/Xew/Animation.h>
Class header file	<X11/Xew/AnimationP.h>
Class	xeAnimationWidgetClass
Class Name	XeAnimation
SuperClass	XeBasic

The Animation Widget class alone does not present any content information, but is a superclass for all “animated” widgets. The Animation Widget will define some common resources for them.

Name	Class	Type	Notes[a]	Default Value
firstFrame	FirstFrame	unsigned	(N)	1
frameCallback	FrameCallback	Callback	(N)	NULL
lastFrame	LastFrame	unsigned long	(N)	Last frame in the sequence
currentFrame	CurrentFrame	unsigned long	(N)	0
doneCallback	DoneCallback	Callback	(N)	NULL
play	Play	XePlay		XePlay_FORWARD

a. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial implementation in some widget classes. "I" in this column indicates resources in which changes will become effective only when the content is loaded/imported into the widget (that is, when **contentLoaded** resource is set to False).

7.1 Defined resources

frameCallback (`XeCR_ANIMATION_FRAME`)

is called the just before the frame shown is equal to the **currentFrame** with the following callback information:

```
typedef struct XeAnimationFrameCallbackData
    {
        int reason;             /* XeCR_ANIMATION_FRAME */
        unsigned long frame;    /* Frame # to be shown */
    } XeAnimationFrameCallbackData;
```

If application wants further frame callbacks, the callback function must change the value of the **currentFrame**.

firstFrame, lastFrame, currentFrame

These resources control the playing of animation sequence. The actual interpretation of what “frame” means is decided in each subclass of this widget. **firstFrame** and **lastFrame** define the sequence that is being played. The **currentFrame** is only used for designating the frame at which the possible **frameCallback** list gets called. The numbering of the frames begins from 1, the value 0 is used to represent unspecified value. Widget may adjust the user supplied values depending on conditions. It may not be possible to start from any frame, and widget will try to choose the nearest possible frame for the real starting point.

doneCallback (`XeCR_ANIMATION_DONE`)
is called when the animation has been completed with the following callback information:

```
typedef struct XeAnimationDoneCallbackData
    {
        int reason;             /* XeCR_ANIMATION_DONE */
        unsigned long frame;    /* Last frame shown */
    } XeAnimationDoneCallbackData;
```

The frame number is zero, if no frames were shown.

play Defines the way how the frames are shown to the user. This resource can have the following values:

XePlay	
Enumeration value (name)	**Converted from String**
`XePlay_FORWARD`	forward
`XePlay_REVERSE`	reverse
`XePlay_FREEZE`	freeze

`XePlay_FORWARD`
play the clip from the **firstFrame** to the **lastFrame**.

`XePlay_REVERSE`
play the clip from the **lastFrame** to the **firstFrame**. (Currently not implemented by any of the subclasses).

`XePlay_FREEZE`
freeze playing at current point.

7.2 Installed converters

The Animation class installs the following resource converters:

from_type	to_type	Description
XtRString	XtRXePlay	Convert string to **XePlay** value

8. Audio Widget

Application header file	<X11/Xew/Audio.h>
Class header file	<X11/Xew/AudioP.h>
Class	xeAudioWidgetClass
Class Name	XeAudio
SuperClass	XeAnimation

An Audio Widget holds the audio content and plays it out when requested. Continous playing is only possible when the stream is passed as file descriptor which is bound to some source (for example unix socket). Audio widget offers an abstract view of the audio: audio is a sequence of consecutive frames (samples) which are numbered from one to infinity[1]. For stereo or multichannel audio each frame may contain two or more samples. Widget will have resources to request playing the audio from any suitable point in sequence (within limits of the selected audio format, not all audio formats allow positioning to every possible frame).

Name	Class	Type	Notes[a]	Default Value
No audio specific resources have been defined at this point.				

a. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial implementation in some widget classes. "I" in this column indicates resources in which changes will become effective only when the content is loaded/imported into the widget (that is, when **contentLoaded** resource is set to False).

8.1 Use of resources from the superclass(es)

The Basic resources **contentFile**, **contentStream** or **contentString** are used to feed the audio content into the widget.

Only **play** is used from Animation superclass (`XePlay_FREEZE` and `XePlay_FORWARD`).

8.2 Translations and actions

Playing control
`toggle-play`
toggle play resource between XePlay_FREEZE and XePlay_FORWARD states.
`restart-play`

1. The frame number is stored into 'unsigned long'. This is at least 32 bit integer and sets an upper limit to frame number as 2^{32}-1. This should be high enough for all practical purposes.

reset input stream to the beginning (achieved by setting the value of **contentLoaded** resource to False. The applicable Basic content* resources must have a valid values, when this is activated.

Miscellaneous

`notify([parameter[,parameter ...]])`

call the Basic **notifyCallback** with reason XeCR_NOTIFY and pass the parameter strings.

The default event bindings for the XeAudio widget are:

```
static char defaultTranslations[] =
"\
<Btn1Down>,<Btn1Up>:            notify()\n\
<Btn1Down>:                     toggle-play()\n\
<Btn3Down>:                     restart-play()\n\
<EnterWindow>:                  enter-window()\n\
<LeaveWindow>:                  leave-window()\n\
";
```

9. Video Widget

Application header file	<X11/Xew/Video.h>
Class header file	<X11/Xew/VideoP.h>
Class	xeVideoWidgetClass
Class Name	XeVideo
SuperClass	XeAnimation

A Video Widget is a rectangular area, that plays video stream when requested or continually. Continous playing is only possible when the stream is passed as file descriptor which is bound to some source (for example unix socket). Video widget offers an abstract view of the video: video is a sequence of consecutive frames which are numbered from one to infinity[1]. Widget will have resources to request playing the video from any suitable point in sequence (within limits of the selected video format, not all video formats allow positioning to every possible frame).

Name	Class	Type	Notes	Default Value
minFrameDelay	MinFrameDelay	int		1

9.1 Defined resources

minFrameDelay

Unless constrained, the current video implementation plays the video as fast as possible. This resource defines the minimum delay between frames in milliseconds.

The value is simply given as parameter to XtAppAddTimeOut function, the timing will be guesswork at best. This resource has nothing to do with timing of frames. The only use for it is to slow down the CPU usage of the widget to allow other X events to be processed.

9.2 Translations and actions

Playing control

`toggle-play`

toggle play resource between XePlay_FREEZE and `XePlay_FORWARD` states.

`restart-play`

reset input stream to the beginning (achieved by setting the value of **contentLoaded** resource to False. The applicable Basic content* resources must have a valid values, when this is activated.

1. The frame number is stored into 'unsigned long'. This is at least 32 bit integer and sets an upper limit to frame number as 2^{32}-1. This should be high enough for all practical purposes.

Miscellaneous
`notify([parameter[,parameter ...]])` call the Basic **notifyCallback** with reason `XeCR_NOTIFY` and pass the parameter strings.

The default event bindings for the XeVideo widget are:

```
static char defaultTranslations[] =
"\
<Btn1Down>,<Btn1Up>:            notify()\n\
<Btn1Down>:                     toggle-play()\n\
<Btn3Down>:                     restart-play()\n\
<EnterWindow>:                  enter-window()\n\
<LeaveWindow>:                  leave-window()\n\
";
```

9.3 Use of resources from the superclass(es)

Many of the imaging related resources of the **Basic** widget are used. They are located in the Basic only because these resources are shared between Video and Raster widgets.

colormapInstall, colormapUse, colorMode, colorQuantize, maxColors
control the colour handing of the images and are supported as described in the Basic documentation.

mirrorImage, rotation, scaling
control the geometry of the image and are supported as described in the Basic documentation.

useShm, visual
are also supported as described in the Basic documentation.

The the following resources from the **Animation** are used

frameCallback, doneCallback
are supported as described in Animation superclass.

currentFrameis used as trigger for frameCallback.

firstFrame, **lastFrame**
specify the first and last frame to be played.

9.4 MPEG format (XeContentFormat_MPEG)

The video widget supports only MPEG 1 format. The implementation is based on the Berkeley MPEG decoder.

10. Frame Widget

Application header file	<X11/Xew/Frame.h>
Class header file	<X11/Xew/FrameP.h>
Class	xeFrameWidgetClass
Class Name	XeFrame
SuperClass	XeBasic

The Frame Widget class is a constraint widget which is modelled after the functionality of the ODA layout object Frame. A Frame widget manages one or more widgets, their distances from each others and their borders. The widget has a concept of *layout path* which can have the four values (0, 90, 180 and 270).

The intention is that the XeFrame behaves according to the ODA specification on parts that are covered by the ODA standards. XeFrame does include features which are not specified in ODA and also allows parameter values outside ODA. If the behaviour of XeFrame differs from ODA specification for equivalent parameter values that are covered by ODA, then this is a bug and will be fixed.

Name	Class	Type	Notes	Default Value
bottomShadowContrast	BottomShadow Contrast	int		40
horizontal	FrameDimension	XeFrameDimension		XeFrameDimension_MAX
invertShadow	InvertShadow	Boolean		False
layoutPath	LayoutPath	int		270
leadingBorder	FrameBorder	XeFrameBorder		No border
leftBorder	FrameBorder	XeFrameBorder		No border
overflowCallback	OverflowCallback	Pointer		NULL
rightBorder	FrameBorder	XeFrameBorder		No border
topShadowContrast	TopShadowContrast	int		20
trailingBorder	FrameBorder	XeFrameBorder		No border
vertical	FrameDimension	XeFrameDimension		XeFrameDimension_MAX

Constraint resources:

Name	Class	Type	Default Value
bottomShadowContrast	BottomShadow Contrast	int	40
centerSeparation	FrameSeparation	int	0
frameAlignment	FrameAlignment	XeFrameAlignment	XeFrameAlignment_RIGHT
invertShadow	InvertShadow	Boolean	False
horizontal	FrameDimension	XeFrameDimension	XeFrameDimension_MAX
leadingBorder	FrameBorder	XeFrameBorder	No border
leadingOffset	FrameOffset	int	int
leadingSeparation	FrameSeparation	int	0
leftBorder	FrameBorder	XeFrameBorder	No borde

Name	Class	Type	Default Value
leftOffset	FrameOffset	int	0
framePosition	FramePosition	XeFramePosition	XeFramePositionType_ NORMAL
rightBorder	FrameBorder	XeFrameBorder	No border
rightOffset	FrameOffset	int	0
topShadowContrast	TopShadowContrast	int	20
trailingOffset	FrameOffset	int	0
trailingBorder	FrameBorder	XeFrameBorder	No border
trailingSeparation	FrameSeparation	int	0
vertical	FrameDimension	XeFrameDimension	XeFrameDimension_MAX

10.1 Defined resources

Some of the resources are both normal XeFrame resources and constraint resources. When a XeFrame object is child of another XeFrame, these resource values will be duplicated both in constraint record and in the child widget itself. The parent XeFrame assumes that the child XeFrame processes some of these. XeFrameBorder type resources are one example of such.

The resource descriptions borrow concepts from the ODA specification. The reader is cautioned to read these definitions carefully. The first thing that can throw a casual reader off the track is that the terms *left* and *right* are defined relative to the l*ayout path* when they refer to the edges. Additionally, the terms *leading* and *trailing* have similar specific meaning.

frameAlignment

defines the desired positioning of the child objects in direction that is orthogonal to the *layout path* (this resource has no effect on objects that have the fixed positioning set by the framePosition resource). The Frame uses only the first three values of the `XeFrameAlignment`:

```
typedef enum
    {
        XeFrameAlignment_LEFT,   /* XeFrame */
        XeFrameAlignment_RIGHT,  /* XeFrame */
        XeFrameAlignment_CENTER, /* XeFrame,XeText */
        XeFrameAlignment_TOP,    /* XeText */
        XeFrameAlignment_BOTTOM, /* XeText */
        XeFrameAlignment_ABOVE,  /* XeText */
        XeFrameAlignment_AROUND, /* XeText */
        XeFrameALignment_BELOW   /* XeText */
    } XeFrameAlignment;
```

`XeFrameAlignment_RIGHT`
: the child object is positioned as close as possible to the right hand edge (as defined by the *layout path*) of then containing frame.

`XeFrameAlignment_LEFT`
: the child object is positioned as close as possible to the left hand edge (as defined by the *layout path*) of the containing frame.

`XeFrameAlignment_CENTER`
: the child object is centered within available area orthogonal to the *layout path* direction.

Because left and right alignments are defined relative to the layout path direction, above definitions must be read with care. The default layout path is 270, and the default alignment is 'right'. These defaults will produce the "normal" top down stacking of children aligned to the ***left*** *as seen by the viewer.*

XeFrame (or XeText) class installs a resource converter for the `XeFrameAlignment` type, which understands the following syntax:

```
<alignment> ::= "left" | "right" | "center" |
                "top" | "bottom" |
                "above" | "around" | "below"
```

framePosition defines the desired positioning of the child objects in *layout path direction*. The resource value is a structure defined as (for the Frame, the possible values are the first three):

```
typedef enum
    {
        XeFramePositionType_NORMAL,  /* XeFrame */
        XeFramePositionType_REVERSE, /* XeFrame */
        XeFramePositionType_FIXED,   /* XeFrame */
        XeFramePositionType_INLINE,  /* XeText */
        XeFramePositionType_START,   /* XeText */
        XeFramePositionType_END      /* XeText */
    } XeFramePositionType;

typedef struct XeFramePosition
    {
        XeFramePositionType type;
        Position x, y;
    } XeFramePosition;
```

`XeFramePositionType_NORMAL`
: the children with *normal fill order* are grouped together and the group

as a whole is placed at the beginning of the frame (beginning is defined relative to the *layout path*)

`XeFramePositionType_REVERSE`
the children with *reverse fill order* are grouped together and the group as a whole is placed at the end of the frame (end is defined relative to the *layout path*.

`XeFramePositionType_FIXED`
the children with *fixed positioning* are placed individually on the specified coordinates (relative to the parent frame) without any consideration for any other children (the objects may overlap other children).

XeFrame (or XeText) class installs a resource converter for the `XeFramePosition` type, which understands the following syntax:

```
<position> ::=  "normal" | "reverse" | <x> ["," <y>]
                "inline" | "start" | "end"
```

Note, that there is no special keyword "fixed"; the request for absolute positioning is implied by the presence of the integer value(s). If either value is omitted, it will default to 0.

For example

```
*FramePosition:normal
*someButton.FramePosition:reverse
*logoImage.FramePosition:10,14
```

horizontal, **vertical (FrameDimension)**
are hints or instructions to the frame about the desired dimensions of the child widget. The resource value is a structure defined as:

```
typedef struct XeFrameDimension
    {
        int value;
        int min, max;
    } XeFrameDimension;
```

The `value` field is interpreted as follows

`value > 0` if greater than zero, this gives the size of the child (constraint resource) or the *preferred size* of the frame self in the specified dimension (horizontal or vertical). The child will not have any choice about the value.

`value == 0 (XeFrameDimension_MAX)`
the dimension is computed by the Frame to be the maximum available size in the specified dimension.

`value < 0` the preferred value for this dimension is asked from the child widget (XeFrame widgets *preferred dimensions* depend on the value field, see below for details).

In all cases, if either field `min` and/or `max` have positive values, they will further constrain the dimension. Further the dimension is always constrained by the size of the frame and by the extra room required by the borders.

If the child widget is a XeFrame and *the dimension is orthogonal to the layout path direction*, the negative values for the `value` field have additional semantics inherited from the ODA specification

`value == -1 (XeFrameDimension_RULE_A)`
the dimension is selected such that it is sufficient for the first child of the child Frame.

`value == -2 (XeFrameDimension_RULE_B)`
the dimension is selected such that all children of the frame will fit in this dimension.

XeFrame (or XeText) class installs a resource converter for the `XeFrameDimension` type, which understands the following syntax:

```
<dimension> ::= <number> |
        ["max"|"rule-a"|"rule-b"]
                ["," <number> ["," <number> ]]
```

For example

```
*FrameDimension:rule-b
```

is a good default. All frames will prefer a size that is sufficiently large to contain all children, and for the non-Frame objects the child preference is used whenever possible.

invertShadow inverts the top and bottom shadow colors (see **topShadowContrast** and **bottomShadowContrast**). This resource has significance only when shadowed borders are in effect.

layoutPath is the direction in which the children within frame will be laid out.

0 from left to right. *right-hand-edge* is on bottom, *left-hand-edge* is on top, t*railing-edge* is on left and *leading-edge* is on right.

90 from bottom to top. *right-hand-edge* is on right, *left-hand-edge* is on left, *trailing-edge* is on bottom and *leading-edge* is on top.

180 from right to left. *right-hand-edge* is on top, *left-hand-edge* is on bottom, *trailing-edge* is on right and *leading-edge* is on left.

270 from top to bottom. r*ight-hand-edge* is on left, *left-hand-edge* is on right, *trailing-edge* is on top and *leading-edge* is on bottom.

leadingBorder, **trailingBorder**, **rightBorder**, **leftBorder (FrameBorder)**

Borders can be individually specified for each edge of the child object of a XeFrame. The borders are drawn *outside (around)* the child object, if it is not XeFrame. If the child object is XeFrame (or subclass), the borders take space from the object self and are drawn *inside* (by the child XeFrame itself).

The resource value is a structure defined as:

```
typedef enum
    {
        XeBorderType_INVISIBLE,
        XeBorderType_SOLID,
        XeBorderType_DASHED,
        XeBorderType_DOT,
        XeBorderType_DASH_DOT,
        XeBorderType_DASH_DOT_DOT,
        XeBorderType_SHADOW
    } XeBorderType;

typedef struct XeFrameBorder
    {
        XeBorderType type;/* Border type selection */
        int width;       /* Width of the border */
        int space;       /* free space */
        unsigned char color[4];
                          /* Border color (RGB), is
                          ** specified if the 4th byte
                          ** is set non-zero.
                          */
    } XeFrameBorder;
```

The 3d shadow border is a quick add-on which does not work properly with the other renditions. `XeBorderType_SHADOW` *should be specified to all four edges, if used.*

XeFrame (or XeText) class installs a resource converter for the `XeFrameBorder` type, which understands the following syntax:

```
<border> ::= <linetype> ["," <width> [","
                 <space>[","<color>]]]]
<linetype> ::= "invisible"|"solid"|"dashed"|"dot"|
                 "dash-dot"|"dash-dot-dot"|"shadow"
```

where `<width>` and `<space>` are given as integer values, and `<color>` a string that is parsable by **XParseColor** function. Omitted numeric values default to 0. Omitted `<color>` defaults to core **border** color for linetypes other than "shadow". A "shadow" linetype defaults to the core **background** color (which then gets adjusted by the **topShadowContrast** and **bottomShadowContrast** resources).

For example

```
*FrameBorder:solid,2,1,red
```

specifies that all borders are to be solid red lines with thickness of 2 pixels, separated from the object by one pixel of free space.

bottomShadowContrast, **topShadowContrast**

resource values are used only if the line bordertype "shadow" has been selected (see also **invertShadow**). The semantics (and resource names) have been directly borrowed from the Xaw3d widgets. Each of the intensities of red, green and blue (rgb) are computed from the reference border color (RGB) using the following formula:

for top shadow color

```
r = (contrast + 100) / 100 * R
g = (contrast + 100) / 100 * G
b = (contrast + 100) / 100 * B
```

and for bottom shadow color

```
r = (100 - contrast) / 100 * R
g = (100 - contrast) / 100 * G
b = (100 - contrast) / 100 * B
```

leftOffset, **rightOffset**, **trailingOffset**, **leadingOffset (FrameOffset)**

control the minimum distances of the child object edges from the frame edges. The distance is counted from the inside edge of the frame border to the outside edge of the child object border.

For example

```
*FrameOffset:2
```

specifies that minimum of 2 pixels of free space must exist between frame border and child border.

leadingSeparation, **centerSeparation**, **trailingSeparation (FrameSeparation)**
control the minimum distance between the child objects from each other within the frame in the *layout path direction*. **trailingSeparation** is the minimum distance from the previous (if any) child in the same **fillOrder**, **leadingSeparation** is the minimum distance from the next (if any) child in the same fill order and **centerSeparation** is the minimum distance between the group of children laid out in normal and reverse fill order.

For example

```
*FrameSeparation:1
```

specifies that at least 1 pixel of free space must exist between each child object in the *layout path direction*.

overflowCallback (`XeCR_FRAME_OVERFLOW`)
is called during the frame layout process, when a child cannot be placed totally within the Frame. The reason field in the callback data has the value `XeCR_FRAME_OVERFLOW`.

```
typedef XeFrameOverflowCallbacData
    {
        int reason;     /* == XeCR_FRAME_OVERFLOW */
        Widget widget;  /* Child Widget */
        XRectangle child; /* Child Widget rectangle */
        XRectangle frame; /* Frame Widget rectangle */
        int redo;       /* Initialized FALSE */
    } XeFrameOverflowCallbackData;
```

If the callback returns without doing anything, the child widget is laid out as proposed, outside the frame rectangle. If the callback function changes the resources of the child and sets the 'redo' field to 'True', the Frame will restart the layout process.

*Requesting relayout must be used carefully. There is nothing that prevents an **overflowCallback** occurring even after the changed resources.*

10.2 Installed converters

The Frame class installs the following resource converters:

from_type	to_type	Description
XtRString	XeRFrameAlignment	Convert string to **XeFrameAlignment** value
	XeRFramePosition	Convert string to **XeFramePosition** structure
	XeRFrameDimension	Convert string to **XeFrameDimension** structure
	XeRFrameBorder	Convert string to **XeFrameBorder** structure

11. Library

The widget set library defines some additional functions which are not directly associated with any widget, but might be usable to applications: generic support functions, color management and font management.

11.1 Support functions

Application header file <X11/Xew/Support.h>

This section describes a collection of support functions which are not directly associated with any widget class.

`XeContentFormat XeFindContentFormat(char *path)`

The function will open the file specified by the `path` parameter and by reading few bytes from the beginning, attempts to guess the content format for the file. Returns `XeContentFormat_UNKNOWN`, if no guess can be made.

`XeContentClass XeFindContentClass(XeContentFormat format)`

The function will take the content `format` parameter and return a content class identifier which can be used in selecting a widget that supports the content format. The content classes are defined by **Support.h** as follows:

XeContentClass	
class symbol	**Widget Class(es)**
`XeContentClass_UNKNOWN`	returned, if format is not yet supported
`XeContentClass_TEXT`	xeTextWidgetClass, xeTextEdWidgetClass
`XeContentClass_RASTER`	xeRasterWidgetClass
`XeContentClass_AUDIO`	xeAudioWidgetClass
`XeContentClass_VIDEO`	xeVideoWidgetClass
`XeContentClass_GEOMETRIC`	none yet

An example use might be the following: If application only requires a display of image from a file, it could use the following code.

```
...
format = XeFindContentFormat("filename.xxx");
if (XeFindContentClass(format)!= XeContentClass_RASTER)
        printf("Unsupported content format\n");
else
        ... create xeRasterWidget ...
...
```

11.2 Color management

Application header file <X11/Xew/Color.h>

The color management is internal library module which hides some of the complexities of X color handling and provides caching and sharing of colors within application.

```
Colormap XeAllocColors(Display *dpy, Window window, Colormap cmap,
            XeColormapUse mode, int max_alloc, int nc,
            XeSample *r, XeSample *g, XeSample *b, Pixel *pixels)
```

Allocate the requested colors (or closest possible) specified by (`r`,`g`,`b`) values from a colormap (`cmap`). This function never returns error, the requested number (`nc`) of pixels is always returned. The return value is the colormap from which the pixels are actually allocated, usually the same that was passed as a parameter. On return, the `pixels` array is filled with the assigned pixels and the (`r`,`g`,`b`) values are updated to the actual values matching these pixels.

The library uses reference counting for the pixels. If the same pixel is assigned to multiple requested colors, the application must also call **XeFreeColors** for that pixel multiple times.

The `max_alloc` (if less than `nc`) can be used to limit the maximum number of distinct colors that can be allocated within this call (effective for `PseudoColor`, `DirectColor` or `GrayScale` visuals). The remaining requested colors are chosen from the already allocated colors.

The mode parameter implements the definition of **`XeColormapUse`** as described in connection of the Basic **colormapUse** resource.

The `window` and `cmap` parameters are coupled, and should be used only as

`cmap == None && window == None`
: create a new colormap using the default visual of the display.

`cmap == None && window != None`
: create a new colormap using the visual of the given window.

`cmap != None && window != None`
: The window is only used in locating the visual in case a creation of another colormap is caused through the `mode` parameter value being either `XeColormapUse_OPTIONAL` or `XeColormapUse_PRIVATE`.

`cmap != None && window == None`

this combination is illegal, if the cmap is not already cached by some other call, because the library needs to know the Visual of the colormap, but there is no way to find it out (default visual is assumed, which may not be correct).

```
Void XeFreeColors(Display *dpy, Colormap cmap, int nc, Pixel *pixels)
```

Release a number (`nc`) of pixels that have been previously allocated by the **XeAllocColors**.

```
void XeQueryColors(Display *dpy, Colormap cmap, int nc,
        Pixel *pixels, XeSample *r, XeSample *g, XeSample *b)
```

Return the RGB (`r`, `g`, `b`) values of the specified number (`nc`) of pixel values. The pixel values (`pixels`) must be valid for the specified colormap, but do not need to be allocated for this application.

11.3 Font management

Application header file <X11/Xew/TextFont.h>

The font management is an internal library which hides some of the complexities of the X font handling and provides matching, caching and sharing of the font structures within application.

```
XFontStruct *XeFindFont(Display *dpy, char *charset, XeFontList font,
        int weight, int posture, int size)
```

Returns a XFontStruct for a font that matches the requested properties.

`charset` is a string defines the requested character set and encoding. This is the two last fields of the XLFD string in lowercase letters (for example, "iso8859-1")

`font` defines the fontlist from which to choose the specific font. The font list is created with **XeMakeFontList**.

`weight` is either normal (0) or bold (2).

`posture` is either upright (0) or italic (1)

`size` is the size of the requested font in pixels (the height)

If none of the existing fonts match the request, the function will attempt to return the default font of the display (usually "`fixed`").

The application must not release the returned value. Returned value is only a pointer to internally cached value.

`XeFontList XeMakeFontList(char *fls, int length)`

parses the font list specification string (`fls`, `length`) and returns a new fontlist identifier. The `fls` need not be a NUL terminated string, only the `length` number of characters are parsed. The `fls` is a comma separated list of X11 font names

<fontlist>	::= <fontstring> ["=" <characterset>] ["," <fontlist>]
<fontstring>	::= <Any X11 font specification mask>
<size>	::= <positive integer>
<characterset>	::= <Any supported X11 characterset name and encoding>
	for example iso8859-1, jisx0201.1976-0, ...

When the **XeFontList** is used in **XeFindFont**, the list is scanned from left until a matching charset is found, or an entry without a charset is reached. Then, the font mask on this entry is used for font selection. The font mask is assumed to have an implicit '*' at the end, thus a mask '-' will match most X11 font names given in XLFD format.

From above, it follows that only the last entry in the font list should be without the character set.

The font list is a component of the Text **fonts** resource, which is composed of multiple font lists coupled with the size attribute.

12. Printing interface

The printing support in Xew is an experimental implementation and a study how a printing based on the Xlib/X Protocol could work in Xt framework. The implementation is supposed to work both with a real X Print server and Bristol Inc's Xprinter.

About the only assumption Xew makes about the printing support is that the printing occurs using standard Xlib calls and there exists something that looks like a Display structure for the printer. Thus, the Xew itself should not need (source) changes for different X Printing solutions. However, different X printing solutions need different setups in the application code. Currently, the following Xlib calls are used by the printing and must be supported by the printing solution:

- all drawing primitives (XDraw...)
- graphic context handling (XCreateGC, XChangeGC, XFreeGC and all GC related convenience functions)
- color management (XCreateColormap, XFreeColormap, XQueryColors, XAllocColor, XFreeColors)
- font management (XListFonts, XLoadQueryFont, XGetFontProperty and XInternAtom)
- image operations (XPutImage)

In Xew the printing is based on Widget. After setting up the Print Server connection, any widget within application can be printed out with single function call (XePrint). For any non-Xew widget, printing tries to print a screen dump of the widget. But, Xew widgets are “printing aware”:

- **XeText**, **XeTextEd** perform a total re-layout using the fonts and metrics provided by the print server, and generate the text directly to the print server. This is not a screen dump, and means that even almost unreadable text on the display is readable on the printout. While printing is in progress, expose events to the original XeText widget content and editing in XeTextEd are disabled (actually, while printing, the content of the original XeText is empty!.
- **XeRaster**, **XeVideo** generate the print image from the current source data (in case of video, this is the current frame being shown).
- **XeFrame** performs a geometry layout using the “printer changed” versions of the child widgets. This means that if the child geometries were changed due to printer requirements, the XeFrame will adjust to them.

When `XePrint` is given a widget to print, it will first re-layout the widget and all its descendants for the printer in depth first order. After this, the top-level widget is tiled into pages, and each page is printed separately by the printing process. A single widget, such as large image, may extend over several pages.

The XeText (or XeTextEd) widget as the top-level widget behaves specially: if the layout width or height (**columnWidth**, **columnHeight**) are not fixed, the layout will use the tiling (page) dimensions for this purpose, and the resulting new size of the Text widget is tiled (instead of the screen version).

```
XePrintProcess XePrint(Widget w, XePrinter *p,
            XePrintPageProc callback, XtPointer client_data)
```

prints the content of widget (`w`) including all children using the printing parameters given in the XePrinter (`p`) structure. The user specified `callback` (**XePrintPageProc**) is called with the `client_data` once before each page (page > 0), and once after all output to the last page has been produces (page == 0).

`XePrint` returns NULL, if all printing completed within this call, and a NON-ZERO opaque printing state pointer, if the printing process is still in progress. The only way to detect the completion of the printing process in such case, is to have the **XePrintPageProc** defined and watch for page==0 call.

Warning! *An application should take care not to start more than one printing for a widget. The Xew is not currently prepared to do multiple simultaneous printouts from the same widget.*

Note! *The current printing implementation always completes within single XePrint call, unless the application chooses to use the XePrintPause in the page callback function.*

The printing parameters are defined as

```
typedef struct XePrinter
  {
      Display *printer;       /* Printer connection */
      Drawable page;          /* Printer page */
      int page_width;         /* Tiling width */
      int page_height;        /* Tiling height */
      Boolean background;     /* Use widget background */
      XeImaging imaging;     /* Imaging parameters */
  } XePrinter;
```

`Display *printer`

is the Display pointer identifying the target printer.

`Drawable page`

is a Drawable that will be used in all Xlib functions that require a drawable. Whether this is needed or not, depends on the printer implementation.

`int page_width, page_height`

define the tiling of the widget being printed (using screen/src units, pixels). Each tile printed as a separate page. If the top level widget being printed is an XeText, and if **columnWidth** or **columnHeight** resource is undefined (has value 0), then the XeText will temporarily substitute page_width and page_height for the corresponding resource value, before doing the layout for the printer.

`Boolean background`

if set 'True', the background is printed. When 'False', the widget background color is set to 'white' and 'foreground' (in Xew) is black (this is most useful for XeText widget, if the application is using white text on dark background, which is not so readable combination on a printed page).

`XeImaging imaging`

are printer imaging controls (this mostly intended to be used if you want to downgrade or override the corresponding widget resources for the printer). For defaults, initialize to all ZERO is sufficient.

```
typedef struct
  {
        Boolean use_shm;
        XeColormapUse colormap_use;
        XeColorMode color_mode;
        XeDither dither;
        XeColorQuantize color_quantize;
        int max_colors;
  } XeImaging;
```

`Boolean use_shm` (**useShm**)

will instruct the printing to utilize MIT shared memory option for images with the X printer. *Do not set this, as none of the current X printing implementations support this!*

`XeColormapUse colormap_use` (**colormapUse**)

the possible values are

- `XeColormapUse_SHARED` (default)
- `XeColormapUse_SHAREDOWN`
- `XeColormapUse_OPTIONAL`

- `XeColormapUse_PRIVATE`

It is somewhat questionable what use this will have on the printer. Not significant, if the print server is in TrueColor mode.

`XeColorMode color_mode` (**colorMode**)
the possible values are

- `XeColorMode_NONE`, use same mode as specified for the widget (default)
- `XeColorMode_MONO`, convert all images to black and white (bitmaps), before sending them to the printer,
- `XeColorMode_GRAY`, convert all color images to grayscale, before sending to the printer,
- `XeColorMode_COLOR`, use colors on printer (basically, send images as they are to the printer, colors included).

`XeDither dither` (**dither**)
the possible values are

- `XeDither_NONE`
- `XeDither_FS4`

and the effect, if any, depends on source image (for example, currently the dither parameter has no effect on palette images).

`XeColorQuantize color_quantize` (**colorQuantize**)
the possible values are

- `XeColorQuantize_FAST` (default)
- `XeColorQuantize_HECKBERT`
- `XeColorQuantize_PPMQUANT`

This has any effect only if the source image is TrueColor and printer shows as PseudoColor. Leave it to the default (0).

`int max_colors` (**maxColors**)
if > 0, defines the maximum number of colours to be used in any single image print. Perhaps useful for fancy effects with “PseudoColor” printers. Has no effect, if the printer is TrueColor. The default (0) makes printing to use the number of colours specified for the widget being printed.

```
void (*XePrintPageProc)(Widget w, XePrintProcess pp,
            int page, XtPointer client_data)
```

is the callback function passed to the `XePrint` function and is called once before every new page (`page > 0`), and once after all output to the last page has been produced (`page == 0`). The widget (`w`) is the original widget being printed. The printing process handle (`pp`) is an opaque data which identifies the internal printing state. This is used, if the callback needs to do some special control actions to the printing process using the other interface functions described later. When the callback is called by the XePrint (or by the printing process it started), the `pp` parameter is always NON-NULL.

The page identifies the current page being printed (if > 0), and the final call after the last page, if == 0. Negative values of page are reserved for future use, and the callback should now just return in such case.

XtPointer `client_data` is the client_data passed to the `XePrint` function.

The task of this callback is to close printing of the previous page (`page > 1`) and prepare for a printing of a new/next page on the printer (`page > 0`). The exact code depends on the X Printing support being used.

Warning! *When the callback is called with page==0, the XePrintProcess pointer (pp) should not be used for any further control calls. If the XePrint function returned with non-NULL value that got stored into some application structure, this callback must ensure that this value is not used any more.*

```
void XePrintCancel(XePrintProcess pp)
void XePrintPause(XePrintProcess pp)
void XePrintResume(XePrintProcess pp)
```

are control functions that can be used from **XePrintPageProc**, or from the application (provided that XePrint returned NON-NULL and XePrintPageProc has not yet been called with `page==0`)

XePrintCancel aborts the printing process as soon as possible. If the **XePrintPageProc** has not yet been called with the `page==0` parameter, the callback occurs when the termination is completed.

XePrintPause suspends the printing process.

XePrintResume resumes the printing process suspended by **XePrintPause**.

13. Error reporting

EuroBridge widget implementations use intrinsics functions **XtAppWarningMsg** and **XtAppErrorMsg** in reporting warnings and errors. These functions are always used through the tailored intermediate functions **XeWidgetWarningMsg** and **XeWidgetErrorMsg** (defined in Basic.c), which supply the fixed information portions to the intrinsics error/warning functions.

The following conventions are followed:

name is always the name of the widget class that is producing the error (XeText, XeRaster, etc..).

type is detailed name of the error chosen by the widget implementation code.

class is always a fixed string **XeWidgetError**.

14. Installation

This section gives generic information about the installation process. The INSTALL.html (or INSTALL.txt) file included with the software will give a more current information.

The widget set distribution consists of several components (more components may be added later).

`Xew-4.0.tar.gz` (required)
: the primary source code distribution contains *two sub-directories*, the basic Xew library source (`Xew`) and a set of example programs (`demo`).

`Xew-4.0-Xp.tar.gz` (optional)
: the source code for the X print server modified for Xew unpacks into Xp sub-directory. The original code is authored by Matthew Reynolds <mreynolds@qualcomm.com>.

`Xew-4.0.ps.gz` (optional)
: the programmers manual (this document) in Postscript form.

`Xew-3.1-HTML.tar.gz` (optional)
: the programmers manual (this document) converted into HTML format. (Not updated for 4.0 at this point).

`Xew-3.1-Miew.tar.gz` (optional)
: the source code of a multimedia *mail viewer*, which can display Internet RFC-822 format messages, and supports also MIME format. This is a more complex example of widgets.

As distributed, Xew does not include libraries for JPEG, TIFF and RLE image formats. To get Xew to support these formats, those libraries must be retrieved separately and installed before compiling the Xew library. Usually any recent version of these libraries will work with the Xew and the versions mentioned below are just examples.

`jpegsrc.v5` JPEG library from the “Independent JPEG Group”. This is a library containing the basic source code for reading and writing JPEG/JFIF. The widget set should always be able to use the latest available version. The principal members of this group are Tom Lane, Philip Gladstone, Luis Ortiz, Lee Crocker, Ge’ Weijers. More information about this program package can be got by addressing the query to jpeg-info@uunet.uu.net.

`tiff-v3.2beta`
TIFF library from Sam Leffler, Silicon Graphics, Inc. This is a library containing the basic source code for reading and writing TIFF files.

`urt-3.1b.tar.Z`
Utah Raster Toolkit is needed for RLE support.

The current widget set implementation compiles with ANSI C or K&R C compiler and under X11 Release 4 or later version of X window system.

14.1 Simple configure and install

The distribution (`Xew-4.0.tar.gz`) comes with GNU autoconf generated `configure` script. The default installation will build Xew, demo and X Print Server (Xp) using the defaults of your system. It will automatically include support for JPEG, TIFF and RLE, if it can find the libraries (from `/usr/local/include` and `/usr/local/lib`, or from the default search path). The default will not use MOTIF subclassing.

Unpack `Xew-4.0.tar.gz` and `Xew-4.0-Xp.tar.gz` in your chosen directory. Each of the packages will unpack into directory "`./Xew-4.0/...`" and will create one or more subdirectories under it.

- `zcat package.tar.gz | tar xvf -`

After unpacking, the simplest configure and installation steps are

1. `cd Xew-4.0`
2. `./configure` or `./configure --with-motif`
3. `make`
4. `make test`

The "`make test`" attempts to run `demo/texted` and X print server (`Xp/pserver`), if compiled. You need X Window DISPLAY to run the test.

If you want to install Xew into your system directories (root required usually), you need to

1. `cd Xew`
2. `make install`
3. `make install.includes` (if '`make install`' did not do this)

14.2 Configure switches

If the simple installation did not give the result and features you require, you need to rerun the configure with additional switches. Use

```
./configure --help
```

to find out about the supported switches in the configure script (this document may not always be quite up to date).

--with-motif[="C compiler options"]

Xew widgets can optionally be compiled as subclassed from the Motif XmManager. This configuration parameter enables the Motif subclassing. The C compiler options are optional and can be used to specify non-standard paths for the Motif includes (-I..) and library (-L..). This switch adds -DUSING_MOTIF_122 definition into Xew builds.

The default is to compile Xew without MOTIF.

--with-local[="C compiler options"]
--with-jpeg[="C compiler options"]
--with-tiff[="C compiler options"]
--with-rle[="C compiler options"]
--without-local
--without-jpeg
--without-tiff
--without-rle

You need to use above flags if the default configuration did not find the libraries (--with), or it did find some, and you do not want to include them (--without).

The use of JPEG, TIFF or RLE packages requires that the following files exist (these lists may change due to version of the library being used, and are just provided as an example)

from jpegsrc.v5 (or later)

- /usr/local/include/jpeglib.h
- /usr/local/include/jerror.h
- /usr/local/include/jconfig.h
- /usr/local/include/jmorecfg.h
- /usr/local/lib/libjpeg.a

from tiff-3.2beta.tar.Z (or later)

- /usr/local/include/tiff.h
- /usr/local/include/tiffio.h
- /usr/local/lib/libtiff.a

from utah raster toolkit

- /usr/local/include/rle.h
- /usr/local/include/rle_code.h
- /usr/local/include/rle_config.h
- /usr/local/include/rle_raw.h

If your includes and libraries were already in /usr/local/include and /usr/local/lib, then the default configure should already have found them. If your common shared includes are in some different place, you can override /usr/local assumption with

```
--with-local="-I<your-include-path> -L<your-library-path>"
```

The '--with-local' can be used with any number of -I and -L options.

If TIFF, JPEG or RLE libraries are installed into non-standard places, you can also specify individually for each of them

```
--with-jpeg="-I<jpeg-include-path> -L<jpeg-library-path>"

--with-tiff="-I<tiff-include-path> -L<tiff-library-path>"

--with-rle="-I<rle-include-path> -L<tiff-library-path>"
```

Additionally, if any of the libraries have non-standard name (other than 'jpeg', 'tiff' or 'rle'), you can also add

```
"-l<library-name>"
```

--with-dolibs[="[-]normal [-]debug [-]profiled [-]shared"]

The template Imakefiles (Imakefile.in) will usually build all libraries (normal, shared, debugger and profiled) which Imake environment indicates being possible. Unfortunately, it quite often happens that regardless of the apparent support, the actual library templates are not working correctly in many installations. For this reason, the configure script will by default override the Imakefile and force only the NormalLib to be built.

By specifying a plain --with-dolibs you can restore the default Imakefile build (configure will not override anything). Or, you can individually enable or disable specific library builds. For example

```
--with-dolibs="-shared"
```

will disable shared library build, but uses the defaults for others.

Note: *Forcing a build of a library type that your environment does not support will not work, and will result compilation problems during the make. Thus --with-dolibs=shared will not magically produce shared libraries, if the system and imake environment does not support them.*

--with-print[="C compiler options"]
--with-xprinter[="C compiler options"]

You can only specify either one of these or none.

Neither (the default)
if the build directory has a subdirectory "./Xp", the configure assumes this directory contains the X Print Server library and client includes, that will be built along with the other things. The definition `-DUSING_PRINTSERVER` is added to the compile flags.

--with-print if the X Print Server library is already installed in some other place. The configure will not test the existence of the ./Xp directory, but will test the existence of the library (with function XSetPrintParams). The definition `-DUSING_PRINTSERVER` is added to the compile flags.

--with-xprinter if you are using the Xprinter printing library from Bristol Technology Inc. The definition `-DUSING_XPRINTER` is added to the compile flags.

--with-xmu[="C compiler options"]
--with-xaw[="C compiler options"]

Athena (Xaw) and Xmu libraries and includes are usually included in the normal X environment, but if you have them installed in a non-standard place, which is not already covered by other configurations (such as --with-local), then you can add the necessary compiler options with these switches.

The build of ./Xew does not use or need Athena, but it does need some Xmu include files.

--with-xew[="C compiler options"]
By default the configure script assumes that the ./demo and other

possible application directories will be using the Xew being built in the current directory. The switch allows you to override this assumption. A simple --with-xew (without any arguments) will just override, and force the application to use the installed Xew from the standard location. Use specific compiler options (-L, -I, -l) to use non-standard locations.

This switch does not affect the ./Xew build process.

--without-shmem
--with-shmem

By default the configure script assumes the support for MIT Shared Memory can be compiled into Xew, if the header <sys/shm.h> exists, and adds `-DSH_MEM` into compile options in that case.

Specifying --without-shmem will prevent this test and does not specify any compile options. This is useful, if you have the header tested by the default configuration, but get compilation problems due to this.

Specifying --with-shmem sets `-DSH_MEM` explicitly without testing anything (not very useful).

--with-debug[="C compiler options"]
--without-debug

Will override CDEBUGFLAGS in demo/Imakefile (and other application files). Normal default for CDEBUGFLAGS is taken from Imake DefaultCDebugFlags. This switch does not affect ./Xew building.

--with-debug sets CDEBUGFLAGS to DebuggableCDebugFlags.

--without-debug
sets CDEBUGFLAGS to OptimizedCDebugFlags (this is what the

DefaultCDebugFlags is set to usually, normally you don't need to use this).

--with-debug="options"
sets CDEBUGFLAGS to the specified options string.

--with-im[="C compiler options"]

The only special input method supported by Xew is BIG5 through something that is called "xcin" (the support for this was provided by You Shung-I <daniel@info1.csie.nctu.edu.tw>). Using this switch adds `-DIM_XCIN` definition into Xew builds. You will probably need to specify additional include and library paths.

--with-cc="cc command"
--with-cdebugflags="C compiler options"

May work as a last ditch solution, if your C compiler and installed Imake environment do not match (typical with "out of box Solaris", which comes configured to use the Sun cc, but which is not present unless you bought it)

Examples

```
./configure     --with-cc=gcc --with-cdebugflags="-O -Wall"
./configure     --with-cc="cc -Aa -D_HPUX_SOURCE" \
                --with cdebugflags=-O
```

14.3 File list from the source distribution

Xew-4.0/...	Description
Xew/AllWidgets.c	Generated automatically by genlist.sh (the script is from MIT Athena distribution).
Xew/AllWidgets.h	Header file for the AllWidgets.c
Xew/Animation.c	Empty skeleton for the XeAnimation Widget
Xew/Animation.h	Public header file of the XeAnimation Widget
Xew/AnimationP.h	Private header file of the XeAnimation Widget
Xew/Audio.c	Empty skeleton for the XeAudio Widget
Xew/Audio.h	Public header file of the XeAudio Widget
Xew/AudioP.h	Private header file of the XeAudio Widget
Xew/AudioLINUX.c	LINUX specific Audio driver for the XeAudio widget (only a sketch and a place holder, not really usable)
Xew/AudioLINUX.h	Private header of the LINUX audio driver.
Xew/AudioSUN.c	SUN specific Audio driver for the XeAudio Widget
Xew/AudioSUN.h	Private header of the SUN audio driver.
Xew/BUGS	A text file listing some bugs that are waiting to be fixed
Xew/Basic.c	Body of the XeBasic widget implementation code
Xew/Basic.h	Public header file of the XeBasic Widget
Xew/BasicColor.h	Private header file for the colour allocation code
Xew/BasicColor.c	Basic colour allocation utilities (partially modified from the xv program)
Xew/BasicP.h	Private header file of the XeBasic Widget
Xew/CHANGELOG	History of changes and revisions made to the widget set
Xew/Color.c	Generic color management for the Xew
Xew/Color.h	Public header file of the color management
Xew/Copyright	Copyright template (text file)
Xew/Frame.c	Body of the XeFrame widget implementation code
Xew/Frame.h	Public header file of the XeFrame Widget
Xew/FrameP.h	Private header file of the XeFrame Widget
Xew/FrameSupport.c	Support functions that are shared between XeFrame and XeText

Xew-4.0/...	Description
Xew/FrameSupport.h	Private header file of the common routines
Xew/GetImage.c	Contains a function to grab a screen dump of the widget window into XImage.
Xew/GetImage.h	Private header file of the GetImage.c
Xew/ImageTools.c	Basic Image processing code (for Raster and Video)
Xew/ImageTools.h	Private header file the basic image processing code
Xew/Imakefile.in	Imakefile template used by the ./configure script to create the Imakefile. Copy this manually into Imakefile and edit, if the ./ configure script does not work for some reason.
Xew/Print.c	Xew printing support functions
Xew/Print.h	Public header file of the printing support
Xew/PrintP.h	Private header file of the printing support
Xew/Raster.c	Main body of the XeRaster Widget implementation, the top level functions of the widget methods (Expose, Initialize, Destroy, etc.)
Xew/Raster.h	Public header file of the XeRaster Widget
Xew/Raster24to8.c	XeRaster Colour Quantification code (modified from the xv program)
Xew/Raster24to8.h	Private header file for the colour quantification module
Xew/RasterGIF.c	XeRaster GIF import module
Xew/RasterGIF.h	Private header file for the GIF import module
Xew/RasterImport.c	XeRaster main program for importing different raster formats, does some common housekeeping and calls the format specific import modules.
Xew/RasterImport.h	Private header file for the import main module
Xew/RasterJPEG.c	XeRaster JPEG/JFIF import module
Xew/RasterJPEG.h	Private header file for the JPEG import module
Xew/RasterLayout.c	XeRaster layout module converts internal image into displayable format (XImage)
Xew/RasterLayout.h	Private header file of the XeRaster layout module
Xew/RasterP.h	Private header of the XeRaster Widget
Xew/RasterPBM.h	Private header file for the PBM import module.
Xew/RasterPBM.c	XeRaster PBM (PBM, PGM and PNM) import module.
Xew/RasterRLE.c	XeRaster RLE import module

Xew-4.0/...	Description
Xew/RasterRLE.h	Private header file for the RLE import module
Xew/RasterTIFF.c	XeRaster TIFF import module
Xew/RasterTIFF.h	Private header file for the TIFF import module
Xew/Support.c	Miscellaneous support functions.
Xew/Support.h	Header of the Support.c
Xew/TODO	Things to do. A free form text file acting as a notepad for things to do in the widget set.
Xew/Text.c	Main body of the XeText Widget implementation, the top level functions of the widget methods (Expose, Initialize, Destroy, etc.)
Xew/Text.h	Public header of the XeText Widget
Xew/TextEd.c	Main body of the XeTextEd Widget implementation.
Xew/TextEd.h	Public header of the XeTextEd Widget
Xew/TextEdP.h	Private header of the XeTextEd Widget
Xew/TextExport.c	XeText support module. Takes care of generating exported octet stream from the text content.
Xew/TextExport.h	Header file of TextExport.
Xew/TextFont.c	XeText fonts handling functions
Xew/TextFont.h	Private header for the text fonts handling
Xew/TextImport.c	XeText main program for importing different text formats (currently imports only ISO 2022/ISO 6429 based streams).
Xew/TextImport.h	Private header for the import main module
Xew/TextLayout.c	XeText layout module formats the text into displayable formats, doing line wrapping and alignments, fixing the positions of the text.
Xew/TextLayout.h	Private header of the XeText layout module
Xew/TextP.h	Private header of the XeText Widget
Xew/VERSION	Version of the release (text file)
Xew/Video.c	Empty skeleton for the XeVideo Widget
Xew/Video.h	Public header for the XeVideo Widget
Xew/VideoImage.c	Prepare video data into XImage structure and display it.
Xew/VideoImage.h	Private header of VideoImage.c
Xew/VideoMPEG.c	Xew main interface module to MPEG decoder

Xew-4.0/...	Description
Xew/VideoMPEG.h	Private header of the MPEG interface
Xew/VideoP.h	Private header for the XeVideo WIdget
Xew/XewInit.c	Copied and modified from the Athena Widget Set
Xew/XewInit.h	Copied and modified from the Athena WIdget Set
Xew/bitmaps/Importfailure	A bitmap that is shown when image import fails (Raster)
Xew/bitmaps/Loudspeaker	A bitmap used by Audio widget (audio open)
Xew/bitmaps/Mutespeaker	A bitmap used by Audio widget (muted audio)
Xew/ccfilter.c	Basic ISO 2022 interpreter implementation.
Xew/ccfilter.h	Private header file of the ccfilter.c
Xew/genlist.sh	Script to generate AllWidgets.c (from the Athena Widget distribution)
mpeg_*	A set of files that have been modified from Berkeley MPEG decoder release 2.0.
Xew/sharedlib.c	Copied and modified from the Athena Widget Set
demo directory	
Demo.ad	A sample application resources file for the test programs.
demo/Imakefile.in	Imakefile template for building the Imakefile. Used by the top level ./configure script. If configure does not work, copy this into Imakefile and edit manually.
demo/audio.c	Create XeAudio widget, and feed the defined content into it.
demo/enriched.tag	Tag specification file for the 'tagged' application. This defines tags that resemble MIME text/enriched specification (the result is not really a usable enriched editor at this point).
demo/fallback.c	Define fallback_resources table to be included into test applications.
demo/frame.c	An example that uses XeFrame (try "frame -d0 file1 file2")
demo/frame_panel.c	Defines UI control panel for XeFrame
demo/html.tag	control file for 'tagged' example (tagged -tags html.tag file)
demo/image_panel.c	Defines UI control panel for XeRaster
demo/printing.c	Application side code to use the Xew printing
demo/printing.h	Header for the printing interface
demo/rtf.h	Header from for rtf_reader.c (used by rtf2Xew.c)

Xew-4.0/...	Description
demo/rtf2Xew.c	An example which converts some trivial RTF into a text stream containing ISO 6429 control sequences (the result can be loaded into XeText)
demo/rtf_reader.c	RTF reader code from the RTF package by Paul Dubois <dubois@primate.wisc.edu> (used by rtf2Xew.c)
demo/simple.c	Display file(s) with Xew widget(s). A very basic widget creation code without any extra frills..
demo/tagged.c	An experiment in using tag support in the XeTextEd widget.
demo/text_panel.c	Defines UI control panel for XeText
demo/texted.c	An application using the XeTextEd widget at low level (gives direct control of ISO 6429 features)
demo/ui_tools.c	User interface elements used by some test programs
demo/ui_tools.h	Header file for user interface elements.
demo/video.c	Create XeVideo widget, and feed the defined content into it.
demo/video_panel.c	Defines UI control panel for XeVideo
demo/viewer.c	Display file(s) with Xew widget(s) using X11R5 porthole/panner widgets or Athena Viewport in X11R4.
xaudio.c, xaudio-README	xaudio program from Andrew Brooks. This allows you to control the audio volume on Sun/SPARC (included because the XeAudio widget self doesn't control the volume).
demo/Text directory (some sample files, the list below is not complete)	
demo/Text/bugs.txt	A text file suitable for texted (contains some known bugs listing).
demo/Text/demo.txt	Test for XeText, use **format:true**
demo/Text/oneko.README	Test for XeText, a README file from oneko program, contains japanese Kanji (iso-2022-jp)
demo/Text/sample.fi	Test for XeText, contains finnish text in ISO 646 national variant.
demo/Text/sample.iso6937	Test for XeText, contains text in ISO 6937
demo/Text/sample.koi	Test for XeText, contains Russian, font registry unknown, force **fonts:13 koi6x13**, for example (cyrillic font, not included in the standard X11 distribution)
demo/Text/sample.ks	Test for XeText, contains Korean text.
demo/Text/sgr.txt	Test for XeText, contains various combinations of SGR controls

15. References

[1] Savela, M., Interchange Formats, EuroBridge Internal Report

[2] Information processing — ISO 7-bit and 8-bit coded character sets — Code extension techniques, ISO 2022 (Third edition 1986)

[3] Information processing — Control functions for 7-bit and 8-bit coded character sets, ISO 6429 (Second edition 1988)

[4] Information processing — Text and office systems — Office Document Architecture (ODA) and interchange format — Part 6: Character content architectures, ISO 8613-6 (1989).

[5] Scheifler, Robert W., Compound Text Encoding — Version 1.1, MIT X Consortium Standard, X Version 11, Release 5.