Project Title:	***EuroBridge***
Project Number:	**RACE 2008**
Deliverable Type:	**I (Internal)**

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

(Version 2.1)

CEC Deliverable Number:	**EB/IREP/VTT/010/F**
Contractual Date Of Delivery:	**-**
Actual Date Of Delivery:	**March 11, 1994**
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 EuroBridge Widget Set	Related Time Period (or Date)	CEC Deliverable
Prepared by Markku Savela	Document Date	Filename Toolkit.book
Checked - Signature	- Date	Approved - Signature - Date

Executive Summary

EuroBridge widget set is a design and implementation of a toolkit for displaying and presenting 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 2.1) 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).

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 describe and specify 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, Compound Document, Hypermedia) will have own widget class. Some of the widgets will be complex and use recursively other basic widgets. All widgets have the common characteristic that they will import the content description as an octet stream from any of the following sources

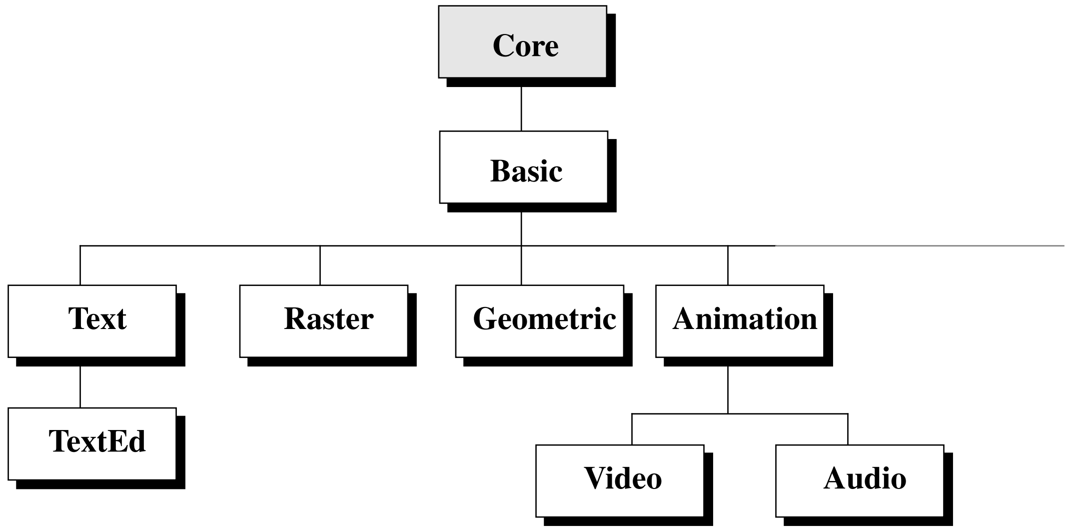
- a named file
- open file descriptor
- pointer to memory

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.

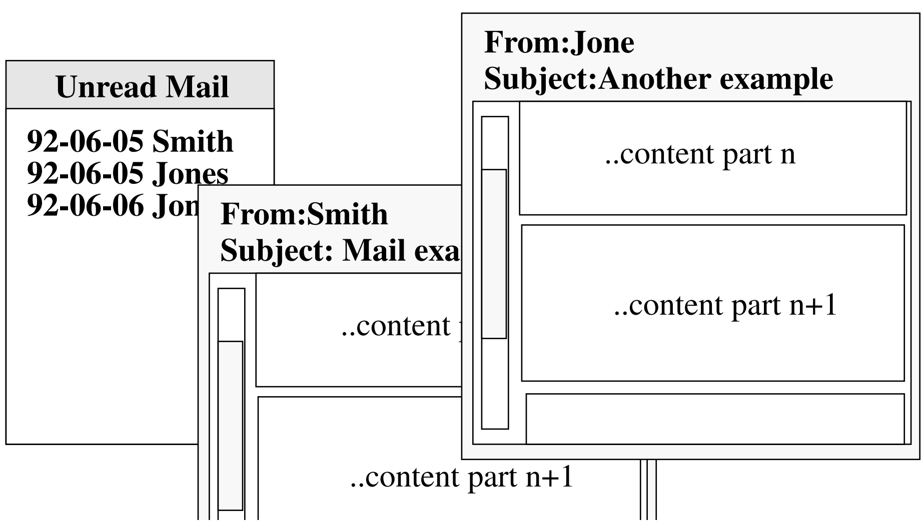
Currently, Text, Raster, Audio and Video have an implementation. This version includes the first editing and cut/paste support for the Text widget. The future versions will include editing and cut/paste support for other widgets also.

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



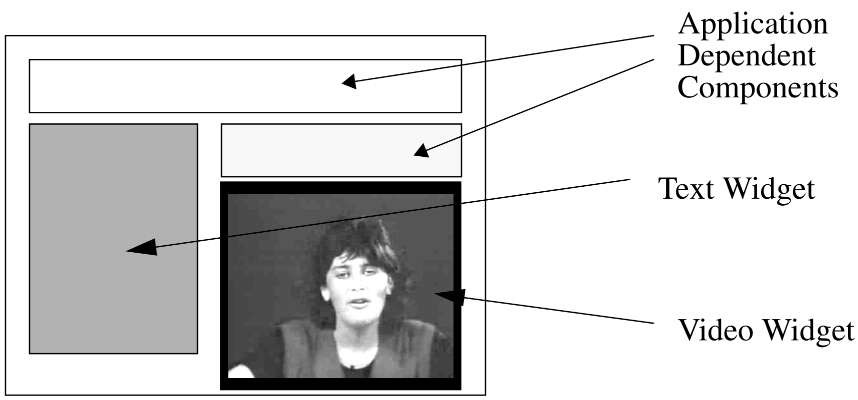
Basic widgets are intended to be used within applications that have a need to present the chosen interchange formats to the user. Examples are

- 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, 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?,...
Hypremedia	?	?

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 one of the modern X Servers on PC/MS-DOS environment are used.

2. Basic Widget

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

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.

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.

Name	**Class**	**Type**	**Notes**[1]	**Default Value**
colormapInstall	ColormapInstall	Boolean		False
colormapUse	ColormapUse	XeColormapUse		XeColormapUse_SHARED
colorMode	ColorMode	XeColorMode		XeColorMode_COLOR
colorQuantize	ColorQuantize	XeColorQuantize		XeColorQuantize_FAST
contentFile	ContentFile	String	I	NULL
contentFormat	ContentFormat	XeContentFormat	I	Default according to widget class.
contentLoaded	ContentLoaded	Boolean		False
contentStream	ContentStream	FILE*	I	NULL
contentString	ContentString	String	I	NULL
contentLength	ContentLength	long	I(N)	Length of the content or end of file
contentOffset	ContentOffset	long	I(N)	0
cursor	Cursor	Cursor		None
exposeCallback	ExposeCallback	Pointer		NULL
foreground	Foreground	Pixel		XtForegroundPixel
maxColors	MaxColors	int		65535
mirrorImage	MirrorImage	Boolean		False
notifyCallback	NotifyCallBack	Pointer		NULL
resize	Resize	Boolean		True

1. "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).

rotation	Rotation	int	0
scaling	Scaling	XeScaling	XeScaling_WIDTH
useShm	UseShm	Boolean	False
visual	Visual	Visual *	NULL

2.1 Defined resources

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**)

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

`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.

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.

*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.***

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**, **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.

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**
XeContentFormat_UNKNOWN	unknown
XeContentFormat_ISO2022	iso2022
XeContentFormat_TIFF	tiff
XeContentFormat_JPEG	jpeg
XeContentFormat_GIF	gif
XeContentFormat_PBM	pbm
XeContentFormat_CGM	cgm
XeContentFormat_H261	h261
XeContentFormat_MPEG	mpeg
XeContentFormat_AUDIO	audio

XeContentFormat	
Enumeration value (name)	**Converted from String**
`XeContentFormat_ODIF`	odif

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**, **contentStream** or **contentString** will be effective. The one used will be the first non-NULL content in order **contentString**, **contentStream**, **contentFile**. 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** and **contentStream** the end is at EndOfFile; for **contentString** the value defaults to the length of NUL-terminated string.

*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.*

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 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.

exposeCallback
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).

maxColors limits the number of colours that will be allocated from the colormap (see **colormapUse**). Restricting colours that may be allocated is advisable in pseudo colour or gray scale 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)

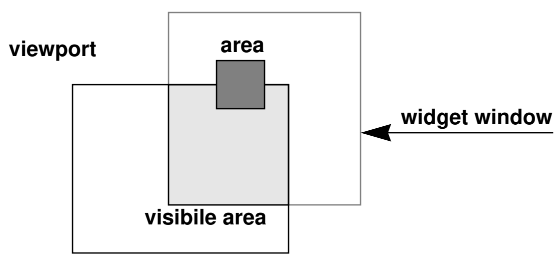
The following two notify callbacks occur via **notifyCallback** and can be used by all widget classes (the detailed method of use may vary from class to class, but the generic idea will be the same).

`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;    /* Event triggering callback */
      String *params;   /* Point 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 the 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.

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.

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

`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\
";
```

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.

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.

Name	**Class**	**Type**	**Notes**[1]	**Default Value**
alignment	Alignment	Alignment		XeAlignment_START
columnWidth	ColumnWidth	Dimension		0
exportFormat	ExportFormat	XeTextExport		XeTextExport_ODIF
firstLineOffset	Offset	int		0
fonts	Fonts	XeFont *	I	Default font as primary font
format	Format	Boolean		False
graphicRendition	Rendition	String	I	Default rendition ("0")
indentation	Offset	Dimension		0
initialState	Sequence	String	I	ISO 8859-1 (G0->LR, G2->GR)
itemization	Itemization	XeItemization *		No itemization
kerningOffset	KerningOffset	Dimension *	N	No kerning offsets
layoutCallback	LayoutCallback	Pointer		NULL
lineLayoutTable	LineLayoutTable	XeTabStop *	N	No tabulation stops for STAB
lineSpacing	LineSpacing	Dimension		Computed from the default font.
proportional	Proportional	Boolean		False

1. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial support. "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).

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 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_CENTER`	center
`XeAlignment_JUSTIFIED`	justified
`XeAlignment_AROUND`	around

columnWidth

when non-zero, this specifies the *linewidth* to be used in laying out the content of the widget. If the value is zero, the *linewidth* is computed from the current **width** of the widget window. This resource is only examined when *folding of lines* is requested (resource **format** has value **True**).

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 3.4 on page 28 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.

fonts describes the fonts that can be used in the text belonging into this widget. This resource is 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.

Currently, the only way to set this resource is through the resource converter. A good default to use is:

**XeText.fonts:12 -*-courier=iso8859-1,-;16;20;24;28;32;36;40;48;60*

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

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>

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. 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 currently recognized character sets that are effective as charset in font list are:

Character Set Names	
Standard reference	**Character Set (X11 name)**
`ISO 8859-1`	iso8859-1

Character Set Names	
Standard reference	**Character Set (X11 name)**
`JIS X0201-1976`	jisx0201.1976-0
`JIS X0208-1983, GL encoding`	jisx0208.1983-0
`KS C5601-1987, GL encoding`	ksc5601.1987-0

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.

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** or from the core resource **width**. If the computed *line width* is zero, no folding of lines occur and the widget behaves like this resource had value False.

graphicRendition

This resource may get removed. The required functionality can be achieved by the initialState resource, when it is allowed to contain ISO 6429 sequences.

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.

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.

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

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 an array of tabulation stop definitions as defined in ODA standard. The end of the array is indicated by an element having NULL value in the `tabulation_reference` field.

```
typedef XeTabStop
 {
      String tabulation_reference;
      Dimension tabulation_position;
      int alignment;
      String alignment_character_string;
 } XeTabStop;
```

The Text class registers a String to XeTabStop* converter[2] with the following syntax with four components, the last of which is optional and only used if the <alignment> = “around”:

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

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.

2. This resource converter has not been written yet.

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 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.

3.3 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.

3.4 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.[3]

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).[4]

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**.

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

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

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.

Ps	Description
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

Ps	Description
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
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**

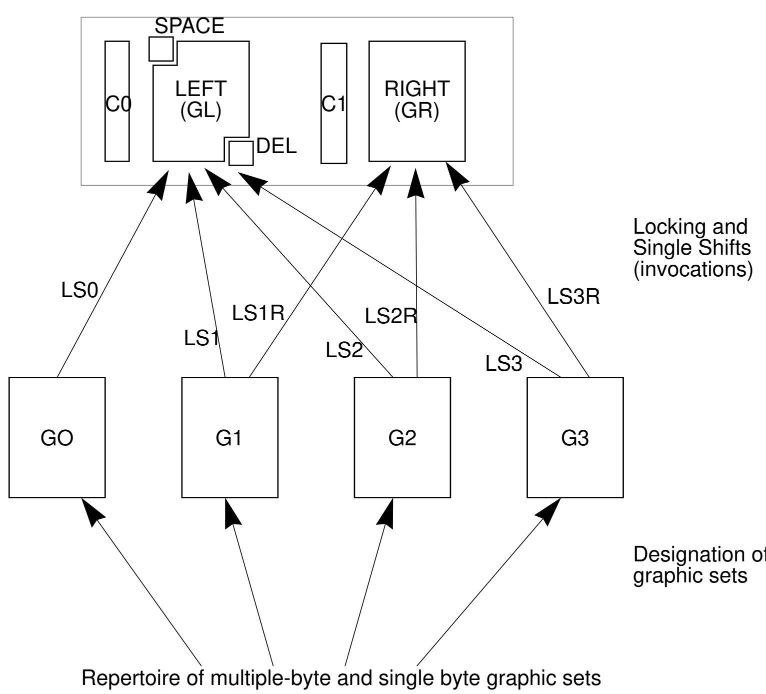
STAB causes subsequent text to be aligned according to the position and the properties of a tabulation stop which is selected from a list according to the value of the parameter Ps.

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[5].



5. 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.

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.

Shift Function	7-bit	8-bit	Invokes characters and side
SHIF-OUT (SO)	0x0e	—	G1 to Left (GL)
SHIFT-IN (SI)	0x0f	—	G0 to Left (GL)
LOCKING-SHIFT ZERO (LS0)	—	0x0f	G0 to Left (GL)
LOCKING-SHIFT ONE (LS1)	—	0x0e	G1 to Left (GL)
LOCKING-SHIFT ONE RIGHT (LS1R)	—	ESC 0x7e	G1 to Right (GR)
LOCKING-SHIFT TWO (LS2)	ESC 0x6e	ESC 0x6e	G2 to Left (GL)
LOCKING-SHIFT TWO RIGHT (LS2R)	—	ESC 0x7d	G2 to Right (GR)
LOCKING-SHIFT THREE (LS3)	ESC 0x6f	ESC 0x6f	G3 to Left (GL)
LOCKING-SHIFT THREE RIGHT (LS3R)	—	ESC 0x7c	G3 to Right (GR)
SINGLE-SHIFT TWO (SS2)	ESC 0x4e	0x8e	Single Character from G2
SINGLE-SHIFT THREE (SS3)	ESC 0x4f	0x8f	Single Character from G3

The current list of supported designations is:

ISO registration number	Character Set	Final character	Mapping to X11 character set encoding (name)
94-character graphic character sets with one intermediate byte			
6	ISO 646, USA Version X3.4 -1968	0x42	iso8859-1
10	ISO 646, Swedish Version, SEN 850200 B	0x47	iso8859-1 (conversion)
11	ISO 646, Swedish Version for Names, SEN 850200 C	0x48	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	jisx0201.1976-0

ISO registration number	Character Set	Final character	Mapping to X11 character set encoding (name)
13	Katakana Character Set JIS C6220-1969	0x49	jisx0201.1976-0
Others	For all unsupported 94 character sets, ISO 8859-1 left hand side is used as an ultimate fallback.	—	iso8859-1
	96-character graphic character sets		
100	Right-hand part of Latin Alphabet No.1, ISO 8859-1, ECMA-94	0x41	iso8859-1
101	Right-hand part of Latin Alphabet No.2, ISO 8859-2, ECMA-94	0x42	iso8859-2
109	Right-hand part of Latin Alphabet No.3, ISO 8859-3, ECMA-94	0x43	iso8859-3
110	Right-hand part of Latin Alphabet No.4, ISO 8859-4, ECMA-94	0x44	iso8859-4
144	Cyrillic part of the Latin/Cyrillic Alphabet?	0x4c	iso8859-5
127	Right-hand part of Latin/Arabic Alphabet ECMA-114, ISO 8859-6	0x47	iso8859-6
126	Right-hand part of Latin/Greek Alphabet, Standard ELOT 928, ECMA-118, ISO 8859-7	0x46	iso8859-7
138	Latin/Hebrew Alphabet, ECMA-121	0x48	iso8859-8
148	Right-hand part of Latin Alphabet No.5	0x4d	iso8859-9
156	Supplementary Set of ISO/IEC 6937	0x52	iso8859-1 (conversion)
Others	For all unsupported 96 character sets, ISO 8859-1 left hand side is used as an ultimate fallback.	—	iso8859-1
	94-character multiple-byte graphic character sets		
58	Chinese Character Set, Chinese Standard GB 2312-80	0x41	gb2312.1980-0
87	Japanese Character Set JIS C 6226-1983	0x42	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	ksc5601.1987-0

ISO registration number	Character Set	Final character	Mapping to X11 character set encoding (name)
Others	Designation ignored, except 2 byte codes assumed and whatever character set was in effect will be used	—	—

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.

New resources

Name	Class	Type	Notes[1]	Default Value
allowEdit	AllowEdit	Boolean		True
cursorPosition	CursorPosition	long		0
displayCaret	Output	Boolean		True
modifyCallback	ModifyCallback	Pointer		NULL

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

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.

1. "N" in this column indicates that the support for the resource has not yet been implemented. "(N)" indicates partial support. "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).

```
typedef struct XeTextModifyCallbackData
 {
      int reason;        /* XeCR_MODIFY_TEXT */
      Boolean cancel;    /* Initialized to False */
 } 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()`

Not implemented.

`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).

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

```
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.

```
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.

```
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 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**.

```
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** function.*

```
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.

```
long XeTextGetInsertionPoint(Widget w)
```

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

```
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 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**.

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

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

```
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[1]	Default Value
clipX	ClipX	long	N	0
clipY	ClipY	long	N	0
clipWidth	ClipWidth	long	N	rasterWidth
clipHeight	ClipHeight	long	N	rasterHeight
offsetX	OffsetX	Dimension		0
offsetY	OffsetY	Dimension		0
rasterWidth	RasterWidth	long		Width from interchange format
rasterHeight	RasterHeigth	long		Height from interchange format

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 will position the upper left corner of the selected rectangle (clip) within the widgets window.

rasterWidth, rasterHeight

specify the actual number of columns and rows of pixels in the raster

1. "N" in this column indicates that the support for the resource has not yet been implemented.

image. Currently, 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.

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 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.

5.5 GIF format (XeContentFormat_GIF)

The GIF format is always supported.

5.6 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.

5.7 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.

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.

The following specification is only a rough sketch to give an idea about possible resources and level of support. The specification may change drastically when a real prototype is implemented.

Name	Class	Type	Notes[1]	Default Value
firstFrame	FirstFrame	unsigned long	(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	N	XePlay_FORWARD

7.1 Defined resources

frameCallback

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

XeCR_ANIMATION_FRAME

```
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**.

1. "N" in this column indicates that the support for the resource has not yet been implemented.

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

is called when the animation has been completed with the following callback information:

XeCR_ANIMATION_DONE

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

The frame # 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.

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[2]	Default Value

No audio specific resources have been defined at this point.

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.

Animation resources?

8.2 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.

Miscellaneous

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

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.
2. "N" in this column indicates that the support for the resource has not yet been implemented.

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[2]	Default Value
minFrameDelay	MinFrameDelay	int		0

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.

2. "N" in this column indicates that the support for the resource has not yet been implemented.

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

f**rameCallback, doneCallback**
are supported as described in Animation superclass.

currentFrameis used as trigger for frameCallback.

9.4 MPEG format (XeContentFormat_MPEG)

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

10. Support

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. 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**.

12. Installation

This section gives only generic information about the installation process. The INSTALL file included with the software will give more current information.

The widget set implementation consists currently of three components (more components may be added later):

- Widget set source code distribution (Xew-2.1.tar.Z). The source is currently available through anonymous FTP from nic.funet.fi:/pub/X11/contrib directory. Information about other possible ways to get the software can be acquired from the EuroBridge project or from the author (savela@tel.vtt.fi).
- 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 library from Sam Leffler, Silicon Graphics, Inc. This is a library containing the basic source code for reading and writing TIFF files.

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. The widget set can be compiled also without JPEG or TIFF support. Basic steps to install the widget set first time

1. Choose a place for Xew and unpack Xew-2.1.tar.Z with

   ```
   zcat Xew-2.1.tar.Z | tar xvf -
   ```

   This will unpack two subdirectories:

   - Xew directory will contain the source of the widget set implementation
   - demo directory will contain some test material and a simple test program.
2. Install the JPEG and TIFF libraries. The widget set needs the JPEG and TIFF libraries installed before it can be compiled. Compile and install packages:

```
- from jpegsrc.v4.tar.Z
      /usr/local/include/jconfig.h
      /usr/local/include/jinclude.h
```

```
        /usr/local/include/jpegdata.h
        /usr/local/lib/libjpeg.a

  - from tiff-3.2beta.tar.Z
        /usr/local/include/tiff.h
        /usr/local/include/tiffio.h
        /usr/local/lib/libtiff.a
```

3. Compile the Xew widget code. The widget set compilation assumes the header files are already installed into their proper place (examine the Xew/Imakefile and modify it to fit your local environment or needs before doing the commands below):

```
cd Xew
xmkmf
make install.includes
make depend
make install
```

4. After this you can try building up the demo/test programs

File list from the source distribution

Xew/	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/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/	Description
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/Copyright	Copyright template (text file)
Xew/Geometric.c	Empty skeleton for the XeGeometric Widget
Xew/Geometric.h	Public header file for the XeGeometric Widget
Xew/GeometricP.h	Private header file for the XeGeometric Widget
Xew/INSTALL	Installation guidelines (text file)
Xew/ImageTools.c	Basic Image processing code (for Raster and Video)
Xew/ImageTools.h	Private header file the basic image processing code
Xew/Imakefile	Imakefile to be used by xmkmf (imake) programs. Copied and modified from the Athena Widget Set.
Xew/Imakefile.dist	Default Imakefile. Copy this into Imakefile and edit to fit your local environment.
Xew/Makefile	Makefile generated by xmkmf (imake)
Xew/README	Generic short introduction what Xew is (text file)
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/	Description
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/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/	Description
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/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/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 very simple sample application resources file for the test programs (this is provided only as an example and starting point, all test programs use the shared fallback resources defined in module fallback.c).
INSTALL	Installation instructions
demo/Imakefile	Imakefile for building the demo and simple test programs
demo/Imakefile.dist	Copy to Imakefile and edit for your local environment.
demo/Makefile	Generated from Imakefile with xmkmf
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/simple.c	Display file(s) with Xew widget(s). A very basic widget creation code without any extra frills..

Xew/	Description
demo/tagged.c	An experiment in using tag support in the XeTextEd widget.
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/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 text files)	
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

13. 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.