Libraries

COLLABORATORS			
	TITLE : Libraries		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 14, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libraries

1.1 Amiga® RKM Libraries: 8 Intuition Images, Line Drawing and Text

```
                Intuition supports two general approaches to creating images,  ←
                   lines, and
text in displays:  through Intuition library calls and through graphics
library calls.

This chapter explains the use of Intuition structures and functions for
creating display imagery.  The Intuition graphical functions provide a
high level interface to the graphics library, giving the application quick
and easy rendering capabilities.  As with any high level calls, some power
and flexibility is sacrificed in order to provide a simple interface.

For more flexibility and control over the graphics, the application can
directly call functions in the graphics library as discussed in the
"Graphics Primitives" chapter.  Intuition also has additional features for
defining custom graphic objects.  See the "BOOPSI" chapter for more
information on these objects.
```


1.2 8 Intuition Images, Line Drawing and Text / Intuition Graphic Objects

```
                Intuition graphic objects are easy to create and economical to use ←
                   .  There
are just three basic types of graphic objects you can use yet these three
types cover most rendering needs:
```

Image
Images are graphic objects that can contain any imagery. They consist of a rectangular bitmap that can be any size and describes each individual pixel to be displayed.

Border
Borders are connected lines of any length and number, drawn ↩ between an arbitrary series of points. They consist of a series of two dimensional coordinates that describe the points between which lines will be drawn.

IntuiText
IntuiText strings are text strings of any length drawn in any ↩ font. They consist of a text string and font specification that describes the text to be rendered.

Each of these three objects may be chained together with other members of the same type. For instance, many lines of text may be rendered as a single object by linking many instances of IntuiText objects together. Only objects of the same type may be linked.

Any of these types can be rendered into any of the Intuition display elements (window, requester, menu, etc.). In fact, the application can often display the same structure in more than one position or more than one of the elements at the same time.

Displaying Images, Borders and IntuiText

Positioning Graphic Objects

1.3 8 / Graphic Objects / Displaying Images, Borders and IntuiText

Images, Borders and IntuiText objects may be directly or ↩ indirectly rendered into the display by the application. The application can draw these objects directly into windows or screens by using one of the functions DrawImage(), DrawBorder() or PrintIText(). The application supplies the appropriate pointer to a Border

,
Image
or
IntuiText
structure
as an argument to the function, as well as position information and a pointer to the correct RastPort. These rendering functions are discussed in more detail below.

The application can also draw these objects indirectly by attaching them to a menu, gadget or requester. As Intuition places these elements on the display, it also renders the associated graphics. The Requester, Gadget, and MenuItem structures contain one or more fields reserved for rendering information. See the specific chapters on these items for information on attaching graphical objects to them.

1.4 8 / Intuition Graphic Objects / Positioning Graphic Objects

The position of these objects is specified as the sum of two independent components: an external component which gives the position of a base reference point for the list of objects, and an internal component which gives the relative offset of a specific object to the base reference point.

The external component is used to position the object list within the display element. For objects drawn indirectly by attaching them to a menu, gadget or requester, this is always a point within the menu, gadget or requester (the top left corner).

For objects drawn directly with the
DrawImage()
,
DrawBorder()
or

PrintIText()
functions, specific x and y coordinates are provided as arguments that specify an offset within the screen's or window's RastPort at which to display the list of objects.

Each object also has an internal, relative component that is added to the external component described above to determine the final position of the object. This allows the application to reuse a graphical object and have it appear relative to each object to which it is attached. For example, if the application has numerous gadgets of the same size, it can use a single
Border
structure to draw lines around all the gadgets. When the gadgets are drawn, the base position of the lines will be taken from each specific gadget in turn.

1.5 8 Intuition Images, Line Drawing and Text / Creating Images

With an Image structure an application can create graphic objects quickly and easily and display them almost anywhere. Images have an additional attribute that makes them even more economical--by changing two simple data elements in the Image structure, the color of the image may be changed.

Images are rectangular bitmaps which individually define the color of each pixel represented. Images may not be masked to allow part of the background to show through. The entire rectangular image is drawn into the target element, overwriting any data it may overlap. All bitplanes defined in the target RastPort within the image's rectangle are overwritten either with image data, ones or zeros.

Images may be directly drawn by the application by using the DrawImage() function, described below. The image may be rendered into any screen or window RastPort with this function. (DrawImageState() can also be used to draw the image.

The visual imagery for an Image can be removed from the display by calling EraseImage(). For a normal Image structure, this will call the graphics function EraseRect(), which clears the Image rectangle by using the layer's backfill pen to overwrite it.

Alternately, images can be used indirectly by attaching them to menus, gadgets or requesters when they are initialized. For instance, in menus the MenuItem structure has the ItemFill and SelectFill fields. If the ITEMTEXT flag is cleared and the HIGHIMAGE flag is set, the application may place a pointer to a list of Image structures in each of these fields. The system will display the ItemFill images when the menu item is not selected and the SelectFill images when the menu item is selected. The application does not have to take any specific action to display these images. Once the menus have been added to a window, their management and display is under Intuition control.

The number of bitplanes in an image does not have to match the number of bitplanes in the display element in which the image is rendered. This provides great flexibility in using Image structures, as the same image may be reused in many places.

If the application's window is on the Workbench or some other public screen, it must use caution with hard-coded or constant image data, as the color palette of that screen is subject to change. If the application has its own custom screen, and it is appropriate for the colors of that screen to change, the same situation arises. Starting with V36, Intuition allows the screen opener to provide a mapping of pen number and rendering functions. For example, pens are specified for the bright and dark edges of three dimensional objects. Applications can obtain this mapping from the DrawInfo structure. See the "Intuition Screens" chapter for more information on DrawInfo and the new 3D look of Intuition in Release 2.

A suitably designed image may be drawn into a screen or window of any depth. To accomplish this, the application must ensure that detail is not lost when the image is displayed in a single bitplane RastPort, where only the first bitplane of image data will be displayed. This is important if the image will ever be displayed on the Workbench screen or any other public screen.

1.6 8 / Creating Images / Image Structure

For images, the application must create one or more instances of the Image structure.

```
struct Image
    {
    WORD LeftEdge;
    WORD TopEdge;
    WORD Width;
    WORD Height;
    WORD Depth;
    UWORD *ImageData;
    UBYTE PlanePick, PlaneOnOff;
    struct Image *NextImage;
    };
```

The meanings of the fields in the Image structure are:

LeftEdge, TopEdge
    The location of the image relative to its base position when it is drawn. These offsets are added to the base position to determine the final location of the image data.

    The base position for images rendered with DrawImage() is taken from arguments passed in the function call. For gadgets and menus, the base position is always the upper, left corner of the select box. For requesters the base position is always the upper, left corner of the requester.

Negative values of LeftEdge and TopEdge move the position up and to the left of the base position. Positive values move down and to the right.

Width, Height
: The width and height of the image. Width contains the actual width of the image in pixels. Height specifies the height of the image in pixels.

The Width field of the Image structure contains the actual width in pixels of the widest part of the image, not how many pixels are contained in the words that define the image.

Depth
: The depth of the image, or the number of bitplanes used to define it. This is not the depth of the screen or window in which the image will be displayed, it is the actual number of bitplanes that are defined in the ImageData.

ImageData
: This is a pointer to the bits that define the image and determine the colors of each pixel. Image data must be placed in Chip memory. The data is organized as an array of 16 bit words whose size can be computed as follows:

```
WordWidth = ((Width + 16) / 16);
NumImageWords = WordWidth * Height * Depth;
```

The width of the image is rounded up to the nearest word (16 bits) and extra trailing bits are ignored. Each line of each bitplane must have enough words to contain the image width, with extra bits at the end of each line set to zero. For example, an image 7 bits wide requires one word for each line in the bitplane, whereas an image 17 bits wide requires two words for each line in the bitplane.

PlanePick
: PlanePick tells which planes of the target BitMap are to receive planes of image data. This field is a bit-wise representation of bitplane numbers. For each bit set in PlanePick, there should be a corresponding bitplane in the image data.

PlaneOnOff
: PlaneOnOff tells whether to set or clear bits in the planes in the target BitMap that receive no image data. This field is a bit-wise representation of bitplane numbers.

NextImage
: This field is a pointer to another instance of an Image structure. Set this field to NULL if this is the last Image structure in the linked list.

1.7 8 / Creating Images / Directly Drawing the Image

As noted above, you use the DrawImage() call to directly draw an image
into a screen or window RastPort.

```
void DrawImage( struct RastPort *rp, struct Image *image,
                long leftOffset, long topOffset );
```

The rp argument is a pointer to the RastPort into which the image should
be drawn. This RastPort may come from a Window or Screen structure.

The image argument is a pointer to the list of
Image
structures that are
to be rendered. The list may contain a single Image structure.

The leftOffset and topOffset arguments are the external component, or the
base position, for this list of images. The
LeftEdge
and
TopEdge
values
of each
Image
structure are added to these values to determine the final
position of each image.

Images may also be indirectly drawn by attaching them to gadgets, menus or
requesters when they are initialized.

1.8 8 / Creating Images / Image Data

Image data must be in Chip memory. The
Image
structure itself may be in
any memory, but the actual data referenced by
ImageData
field must be in
Chip memory. This may be done by using compiler specific options, such as
the __chip keyword of SAS/C, or by allocating memory with the MEMF_CHIP
attribute and copying the image data to that memory.

Defining Image Data

1.9 8 / / Image Data / Defining Image Data

Image data consists of binary data organized into a series of 16-bit words. The words must be sequential, where each successive word represents bits that are displayed later in the image. The image is defined as follows:

* The image is broken down into bitplanes. Each bitplane is considered separately.

* Within a single bitplane, each row of pixels is taken separately. First, round the number of pixels up to the next even multiple of 16. This determines the number of words used to represent a single row of data. For instance, an image that is 17 bits wide will require two 16-bit words to represent each row.

  The leftmost 16 pixel values are placed in the first word, followed by the next 16 pixel values, and so on. Any extra pixels at the end of the last word of the
  ImageData
  should be set to zero.

* The first row of data is the topmost row of the low order bitplane. This is immediately followed by the second row, then the third, until all rows in the bitplane have been represented.

* The data for the low order bitplane is followed immediately by the next to lowest, then the next, etc.

The color of each pixel in the image is directly related to the value in one or more memory bits, depending upon how many bitplanes there are in the image data and in which bitplanes of the screen or window the display is displayed.

The color for a single pixel may be determined by combining the bits taken from the same relative position within each of the bitplanes used to define the image. For each pixel, the system combines all the bits in the same position to create a binary value that corresponds to one of the system color registers. This method of determining pixel color is called color indirection, because the actual color value is not in the display memory. Instead, it is in color registers that are located somewhere else in memory.

In many situations, the image and display will have different number of bitplanes, which complicates the determination of the color value for a given pixel. For now, assume that the image and display have the same number of bitplanes. The more complex example will be covered below, in the section "
Image Example
".

If an image consists of only one bitplane and is displayed in a one bitplane display, then wherever there is a 0 bit in the image data, the color in color register zero is displayed and wherever there is a 1 bit, the color in color register one is displayed.

In an image composed of two bitplanes, the color of each pixel is obtained from a binary number formed by the values in two bits, one from the first bitplane and one from the second bitplane. If the bit in the first bitplane is a 1 and the bit in the second bitplane is a 0, then the color of that pixel will be taken from color register two (since 10 in binary is two in decimal). Again, the first bitplane describes all of the low order bits for each pixel. The second bitplane describes the next higher bit, and so on. This can be extended to any number of bitplanes.

```
               Image Data           Hexadecimal Representation

    ***************************········    F F F F    F F 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    **·······················**········    C 0 0 0    0 3 0 0
    ***************************········    F F F F    F F 0 0
```

Figure 8-1: Rendering of the Following Example Image

simpleimage.c

1.10 8 / Creating Images / Picking Bitplanes for Image Display

A single image may be displayed in different colors without changing the underlying image data. This is done by selecting which of the target bitplanes are to receive the image data, and what to do with the target bitplanes that do not receive any image data.

PlanePick and PlaneOnOff are used to control the bitplane rendering of the image. The bits in each of these variables have a direct correspondence to the bitplanes of the target bitmap. The lowest bit position corresponds to the lowest numbered bitplane, the next highest bit position corresponds to the next bitplane, etc.

For example, for a window or screen with three bitplanes (consisting of planes 0, 1, and 2), all the possible values for PlanePick or PlaneOnOff and the planes picked are as follows:

```
 __________________________________
|                                  |
| PlanePick or                     |
|  PlaneOnOff   Planes Picked      |
```

```
| ------------  -------------      |
|     000       No planes          |
|     001       Plane 0            |
|     010       Plane 1            |
|     011       Planes 0 and 1     |
|     100       Plane 2            |
|     101       Planes 0 and 2     |
|     110       Planes 1 and 2     |
|     111       Planes 0, 1, and 2 |
|__________________________________|
```

PlanePick
picks the bitplanes of the containing RastPort that will receive the bitplanes of the image. For each plane that is picked to receive data, the next successive plane of image data is drawn there. For example, if an image with two bitplanes is drawn into a window with four bitplanes with a PlanePick of binary 1010, the first bitplane of the image will be drawn into the second bitplane of the window and the second bitplane of the image will be drawn into the fourth bitplane of the window. Do not set more bits in PlanePick than there are bitplanes in the image data.

PlaneOnOff
specifies what to do with the bitplanes that are not picked to receive image data. If the PlaneOnOff bit is zero, then the associated bitplane will be filled with zeros. If the PlaneOnOff bit is one, then the associated bitplane will be filled with ones. Of course, only bits that fall within the rectangle defined by the image are affected by this manipulation.

Only the bits not set in
PlanePick
are used in
PlaneOnOff
, that is,
PlaneOnOff only applies to those bitplanes not picked to receive image data. For example, if PlanePick is 1010 and PlaneOnOff is 1100, then PlaneOnOff may be viewed as x1x0 (where the x positions are not taken into consideration). In this case, planes two and four would receive image data and planes one and three would be set by PlaneOnOff. Each bit in plane one would be set to zero and each bit in plane three would be set to one.

PlaneOnOff
is only useful where an entire bitplane of an image may be set to the same value. If the bitplane is not all set to the same value, even for just a few bits, then image data must be specified for that plane.

A simple trick to create a filled rectangle of any color may be used by supplying no image data, where the color is controlled by
PlaneOnOff
. The

Depth
 of such an image is set to zero, the size of the rectangle is
specified in the
Width
 and
Height
 fields and the
ImageData
 pointer may be
NULL. PlanePick should be set to zero, as there are no planes of image
data to pick. PlaneOnOff is then set to the color register which contains
the desired color for the rectangle.

1.11 8 / Creating Images / Image Example

A more complex example of the use of an
Image
 is presented below.

```
     Plane 0, Open Rectangle

  ************************········
  **····················**········
  **····················**········
  **····················**········
  **····················**········
  **····················**········
  **····················**········
  **····················**········
  **····················**········
  ************************········

    Plane 1, Filled Rectangle

  ································
  ································
  ································
  ········oooooooo················
  ········oooooooo················
  ········oooooooo················
  ········oooooooo················
  ································
  ································
  ································

      3-Color Combined Image

  ************************········
  **····················**········
  **····················**········
  **······oooooooo······**········
  **······oooooooo······**········
  **······oooooooo······**········
```

```
**......oooooooo......**........
**....................**........
**....................**........
************************........
```

Figure 8-2: Picture of the More Complex Example Image

complexeimage.c

1.12 8 Intuition Images, Line Drawing and Text / Creating Borders

This data type is called a Border since it was originally used to create border lines around display objects. It is actually a general purpose structure for drawing connected lines between any series of points.

A Border is easier to use than an Image structure. Only the following need be specified to define a border:

* An internal position component which is used in determining the final position of the border.
* A set of coordinate pairs for each vertex.
* A color for the lines.
* One of several drawing modes.

Border Structure Definition

Border Colors and Drawing Modes

Directly Drawing the Borders

Border Coordinates
Border Example
Linking Borders

1.13 8 / Creating Borders / Border Structure Definition

To use a border, the application must create one or more instances of the

Border structure. Here is the specification:

```
    struct Border
        {
        WORD LeftEdge, TopEdge;
        UBYTE FrontPen, BackPen;
        UBYTE DrawMode;
        BYTE Count;
        WORD *XY;
        struct Border *NextBorder;
        };
```

Here is a brief description of the fields of the Border structure.

LeftEdge, TopEdge
    These fields are used to determine the position of the Border relative to its base position (the base position is the upper left corner for requesters, menus, or gadgets and is specified in the call to
    DrawBorder()
    for windows and screens).

FrontPen, BackPen
    These fields contain color registers numbers. FrontPen is the color used to draw the lines. BackPen is currently unused.

DrawMode
    Set the DrawMode field to one of the following:

    JAM1
        Use FrontPen to draw the line.

    COMPLEMENT
        Change the pixels within the lines to their complement color.

Count
    Specify the number of data points used in this border. Each data point is described by two words of data in the XY array.

XY A pointer to an array of coordinate pairs, one pair for each point. These coordinates are measured relative to the position of the border.

NextBorder
    This field is a pointer to another instance of a Border structure. Set this field to NULL if this is the last Border structure in the linked list.

1.14 8 / Creating Borders / Directly Drawing the Borders

Borders may be directly drawn by the application by calling the ↩ function
DrawBorder().

```
    void DrawBorder( struct RastPort *rp, struct Border *border,
```

```
                     long leftOffset, long topOffset );
```

The rp argument is a pointer to the RastPort into which the border should be drawn. This rastport may come from a Window or Screen structure.

The border argument is a pointer to a list of
Border
structures which are
to be rendered. The list may contain a single Border structure.

The leftOffset and topOffset arguments are the external component, or base position, for this list of
Border
s. The
LeftEdge
and
TopEdge
values of
each Border structure are added to these to determine the Border position.

Borders may also be indirectly drawn by attaching them to gadgets, menus or requesters.

1.15 8 / Creating Borders / Border Colors and Drawing Modes

Borders can select their colors from the values set in the color ↩
registers
for the screen in which they are rendered. The available number of colors and palette settings are screen attributes and may not be changed through border rendering.

Two drawing modes pertain to border lines:
JAM1
and
COMPLEMENT
. To draw
the line in a specific color, use the JAM1 draw mode. This mode converts each pixel in the line to the color set in the
FrontPen
field.

Selecting the
COMPLEMENT
draw mode causes the line to be drawn in an
exclusive-or mode that inverts the color of each pixel within the line. The data bits of the pixel are changed to their binary complement. This complement is formed by reversing all bits in the binary representation of the color register number. In a three bitplane display, for example, color 6 is 110 in binary. In COMPLEMENT draw mode, if a pixel is color 6, it will be changed to the 001 (binary), which is color 1. Note that a border drawn in COMPLEMENT mode can be removed from a static display by drawing the border again in the same position.

1.16 8 / Creating Borders / Border Coordinates

Intuition draws lines between points that are specified as sets of X, Y coordinates. Border data does not have to be in Chip memory.

The XY field contains a pointer to an array of coordinate pairs. All of these coordinates are offsets relative to the Border position, which is determined by the sum of the external and internal position components as described above. The coordinate pairs are ordered sequentially. The first two numbers make up the first coordinate pair, the next two numbers make up the second pair, and so on. Within a coordinate pair, the first number is the X offset and the second number is the Y offset.

The first coordinate pair describes the starting point of the first line. When the Border is rendered, a line is drawn between each pair of points. The first line is drawn from point one to point two, the second line is drawn from point two to point three, and so on, until the final point is reached.

The numbers specified in the XY array may be positive or negative. Negative values move up and to the left relative to the Border position, positive values move down and to the right. Again, the Border position is determined by adding the external position component and the internal position component. For example, a Border attached to a Gadget has an external component equal to the upper left corner of the gadget's select box. The internal component is set within the Border structure itself. These two components are added together and offsets from the resulting position, specified within the XY array, determine where the lines of the Border will appear.

Suppose the top left corner of the select box of the gadget is at window position (10,5). If the Border has LeftEdge set to 10 and TopEdge set to 10, then the Border is positioned at (10+10,5+10), that is (20,15). All XY coordinates will be relative to this Border position. If the XY array contains ‘0,5, 15,5, 15,0’, then the relative coordinates will be (0,5), (15,5) and (15,0). Adding each coordinate to the Border position gives the absolute position of the lines within the window. This Border will draw two lines in the window, one from (20,20) to (35,20) and the second from (35,20) to (35,15).

```
          0    5   10   15   20   25   30   35   40   45   50
          |         |         |         |         |         |
       0__|____|____|____|____|____|____|____|____|____|____|
          |   Top left corner of the
          | gadget's select box (10,5)
       5 _|         * _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
          |
          |         |
      10__|                Border          Third
          |         |     Position       Coordinate
          |                (20,15)      (20+15,15+0)
      15 _|         |         *               *
          |                                   |
          |         |                         |
      20__|                   *_______________*
          |         |       First          Second
          |               Coordinate      Coordinate
      25 _|         |     (20+0,15+5)    (20+15,15+5)
          |
          |         |
      30__|
```

Figure 8-3: Example of Border Relative Position

To create a border that is outside the select box of a gadget, specify negative values in the internal component or use negative values for the initial XY values. For example, setting LeftEdge to -1 and TopEdge to -1 moves the position of the Border one pixel above and one pixel to the left of the gadget’s select box.

1.17 8 / Creating Borders / Linking Borders

The NextBorder field can point to another instance of a Border structure.

This allows complex graphic objects to be created by linking together Border structures, each with its own data points, color and draw mode. This might be used, for instance, to draw a double border around a requester or gadget where the outer border is a second Border structure, linked to the first inner border.

Note that the borders can share data. For instance, to create a border with a shadow, link two borders together each of which points to the same

XY
data. Set the first border to draw in a dark pen (such as the SHADOWPEN from the screen's DrawInfo structure) and position the border down and to the right a few pixels by changing
LeftEdge
and
TopEdge
in the

Border
structure.

The second border should be set to a bright pen (such as the SHINEPEN in the screen's DrawInfo structure). When the border is drawn, the first border will draw in a dark color and then the second border will be drawn over it in a light color. Since they use the same data set, and the dark border is shifted down and to the right, the border will have a three dimensional appearance. This technique is demonstrated in the example listed earlier in this section.

1.18 8 Intuition Images, Line Drawing and Text / Creating Text

The
IntuiText
structure provides a simple way of writing text strings within an Intuition display element. These strings may be used in windows, screens, menus, gadgets and requesters. To set up an IntuiText, you specify the following:

* Pen colors for the text.

* A draw mode.

* The starting offset for the text.

* The font used to render the text.

* The text string to output.

IntuiText Structure

Text Colors and Drawing Modes

Directly Drawing the IntuiText

1.19 8 / Creating Text / IntuiText Structure

To render text using Intuition, the application must create one or more
instances of the IntuiText structure:

```
struct IntuiText
    {
    UBYTE FrontPen, BackPen;
    UBYTE DrawMode;
    WORD LeftEdge;
    WORD TopEdge;
    struct TextAttr *ITextFont;
    UBYTE *IText;
    struct IntuiText *NextText;
    };
```

Here is a brief description of each member of the IntuiText structure:

FrontPen
    The pen number specifying the color used to draw the text.

BackPen
    The pen number specifying the color used to draw the background for
    the text, if JAM2 drawing mode is specified.

DrawMode
    This field specifies one of four drawing modes:

    JAM1
        FrontPen is used to draw the text; background color is unchanged.

    JAM2
        FrontPen is used to draw the text; background color is changed
        to the color in BackPen.

    COMPLEMENT
        The characters are drawn in the complement of the colors that
        were in the background.

    INVERSVID
        Inverses the draw modes describe above. For instance INVERVID
        used with JAM1 means the character is untouched while the
        background is filled with the color of the FrontPen.

LeftEdge and TopEdge

The location of the text relative to its base position when it is drawn. These offsets are added to the base position to determine the final location of the text data.

The base position for text rendered with PrintIText() is taken from arguments passed in the function call. For gadgets and menus, the base position is always the upper, left corner of the select box. For requesters the base position is always the upper, left corner of the requester.

LeftEdge gives the offset of the left edge of the character cell and TopEdge gives the offset of the top edge of the character cell for the first character in the text string. Negative values of LeftEdge and TopEdge move the position up and to the left of the base position. Positive values move down and to the right.

ITextFont
A pointer to a TextAttr structure defining the font to be used. Set this to NULL to use the default font.

IText
A pointer to the NULL terminated text string to be displayed.

NextText
A pointer to another instance of IntuiText. Set this field to NULL for the last IntuiText in a list.

1.20 8 / Creating Text / Directly Drawing the IntuiText

Use the PrintIText() call to directly draw the text into the target RastPort of a window or screen.

```
void PrintIText( struct RastPort *rp, struct IntuiText *iText,
                 long left, long top );
```

The rp argument is a pointer to the RastPort into which the text should be drawn. This RastPort can come from a Window or Screen structure.

The iText argument is a pointer to a list of IntuiText structures which are to be rendered. The list may contain a single IntuiText structure. If the font is not specified in the IntuiText structure, Intuition will render the text using the RastPort's font.

The left and top arguments give the external component, or base position for this list of IntuiText structures. The LeftEdge and

TopEdge
values in
each IntuiText structure are added to these to determine the final
position of the text.

IntuiText
objects may also be drawn indirectly by attaching them to
gadgets, menus or requesters.

1.21 8 / Creating Text / Determining Text Length

To determine the pixel length of a given
IntuiText
string, call the
IntuiTextLength() function.

```
    LONG IntuiTextLength( struct IntuiText *iText );
```

Set the iText argument to point to the
IntuiText
structure whose length is
to be found. This function will return the length of the iText text
string in pixels. Note that if the
ITextFont
field of the given IntuiText
is set to NULL, or Intuition cannot access the specified font, then
GfxBase->DefaultFont will be used in determining the length of the text.
This may not be the same as the RastPort font with which the text would be
printed.

1.22 8 / Creating Text / Text Colors and Drawing Modes

IntuiText
gets its colors from the values set in the color registers for
the screen in which they are rendered. The available number of colors and
palette settings are screen attributes and cannot be changed through
IntuiText rendering.

Text characters in general are made up of two areas: the character image
itself and the background area surrounding the character image. The color
used in each area is determined by the draw mode which can be set to
JAM1
,

JAM2
or
COMPLEMENT
. The flag

INVERSVID
may also be specified.

JAM1
draw mode renders each character with
FrontPen
and leaves the
background area unaffected. Because the background of a character is not drawn, the pixels of the destination memory around the character image are not disturbed. Graphics beneath the text will be visible in the background area of each character cell.

JAM2
draw mode renders each character with
FrontPen
and renders each
character background with
BackPen
. Using this mode, any graphics that
previously appeared beneath the character cells will be totally overwritten.

COMPLEMENT
draw mode renders the pixels of each character as the binary complement of the color that is currently at the destination pixel. The destination is the display memory where the text is drawn. As with
JAM1
,
nothing is drawn into the background.
FrontPen
and
BackPen
are not used
in COMPLEMENT mode. To determine the complement color, invert all the bits in the binary representation of the color register number. The resulting number specifies the color register to use for that pixel. In a three bitplane display, for example, color 6 (110 in binary) is the complement of color 1 (001 in binary).

The
INVERSVID
flag inverses the video for each of the drawing modes. For

JAM1
, nothing is drawn into the character area and the background is drawn
in
FrontPen
. For
JAM2
, the character area is drawn in
BackPen
and the
background is drawn in FrontPen. For

COMPLEMENT
mode, nothing is drawn
into the character area and the background is complemented.

1.23 8 / Creating Text / Fonts

The application may choose to specify the font used in rendering the

IntuiText
, or it may choose to use the default font for the system.

To use the default font, set the
ITextFont
field to NULL. Some care must
be taken when using the default font. When an
IntuiText
object is
rendered and no font is specified, the text will be rendered in the font
set in the RastPort.

If the RastPort font is NULL, the text will be rendered using
GfxBase->DefaultFont. Also,
IntuiTextLength()
always uses
GfxBase->DefaultFont when
ITextFont
is NULL. The application must have
open the graphics library in order to check the default font in GfxBase.
(See the graphics library chapter for more information.)

To use a specific font for this text, place a pointer to an initialized
TextAttr structure in the
ITextFont
field. Intuition will only use the
specified font if it is available through a call to the OpenFont()
routine. To use a font from disk, the application must first open the
font using the OpenDiskFont() function. For more information about using
fonts, see the "Graphics Library and Text" chapter in this manual.

1.24 8 / Creating Text / Linking Text Strings

The
NextText
field can point to another instance of an
IntuiText
structure. This allows the application to create a complex object which
has several distinct groups of characters, each with its own color, font,
location, and drawing mode. This can be used to create multiple lines of

text, to position characters in the text very accurately and to change the color or font of the text. Each list of IntuiText objects may be drawn with one call to
PrintIText()
, or attached to a gadget, menu or requester
as a single object.

1.25 8 Intuition Images, Line Drawing and Text / Function Reference

The following are brief descriptions of the Intuition functions that relate to the use of graphics under Intuition. See the Amiga ROM Kernel Reference Manual: Includes and Autodocs for details on each function call.

Table 8-1: Functions for Intuition Drawing Capabilities

Function	Description
DrawBorder()	Draw a border into a rast port.
DrawImage()	Draw a image into a rast port.
PrintIText()	Draw Intuition text into a rast port.
IntuiTextLength()	Find the length of an IntuiText string.
BeginRefresh()	Begin optimized rendering after a refresh event.
EndRefresh()	End optimized rendering after a refresh event.
GetScreenDrawInfo()	Get screen drawing information (V36).
FreeScreenDrawInfo()	Free screen drawing information (V36).