Standard Graphic Element Types

The Graphic Elements system includes a number of "standard" Graphic Element types. All of these elements share certain common characteristics. Each type of Graphic Element has its own creation routine, and several of the standard types provide additional procedures which can be used only with elements of that type.

Characteristics Common to All Graphic Elements

All newly-created Graphic Elements are visible, and will be shown the next time the application program calls DoWorldUpdate() unless the application program makes them invisible. The application program (or autochange procedure) can change the position, visibility, or plane of any element of any type using the display controller routines described in the last section.

The creation routines for all of the standard Graphic Elements have a mode parameter, representing the graphic transfer mode to be using in drawing the element. This mode should be either srcCopy or transparent for all Graphic Elements based on offscreen pixel maps, i.e., all standard elements except the Text Element. The mode should usually be transparent for all such elements except those that make up the "bottom-most" background of the display and those which are converted to "masked" graphics . For Text Elements only, the mode parameter should normally be equal to srcOr.

[Masked Graphics … no documentation yet]

Picture Elements and Tiled Elements

The Picture Element is the simplest type of Graphic Element. It is based on a single image — on the Macintosh, normally a PICT resource — and its appearance never changes. A new Picture Element is created and added to a GEWorld by calling:

```
GrafElPtr NewBasicPICT(GEWorldPtr world, OSType id,
                       short plane, short resNum, short mode,
                       short xPos, short yPos);
```

Where:

— world is the GEWorld into which the new Graphic Element is to be installed,

— id is the unique four-character name by which the application program will refer to this Graphic Element,

— plane is this element's "height" above the background,

— resNum is the resource number of the PICT resource that is the source for this element,

— mode is the graphic transfer mode used to draw this element, normally srcCopy for the bottom-most Graphic Element(s) in the scene and elements for which "masks" have been made, and transparent for the rest,

— xPos and yPos are the horizontal and vertical positions of the upper left-hand corner of this element in its world.

Like a Picture Element, a Tiled Element is normally based on a single PICT resource. It differs in that a destination rectangle is specified at its creation, and its rendering procedure draws it as many times as necessary to fill this rectangle horizontally and vertically. A Tiled Element is created by calling:

```
GrafElPtr NewTiledGraphic(GEWorldPtr world, OSType id,
                          short plane, short resNum,
                          short mode, Rect destRect);
```

Where:

— world, id, plane, resNum, and mode are as defined above for the Picture Element,

— destRect is the rectangle to be tiled with this element, in the coordinates of its GEWorld.

Scrolling Graphic Elements

Like a Picture Element, a Scrolling Element is normally based on a single PICT resource. The Scrolling Element can scroll, horizontally or vertically or both, within its display rectangle. Scrolling Elements automatically "wrap around," so that the top follows the bottom, the left side follows the right side, etc. The application program creates a Scrolling Element by calling:

```
GrafElPtr NewScrollingGraphic(GEWorldPtr world, OSType id,
                              short plane, short resNum,
                              short mode, short xPos, short yPos);
```

Where the meanings of the parameters are the same as for NewBasicPICT(), above.

The autochange procedure of a Scrolling Element (or the application program) sets its current scroll position by calling:

```
void SetScroll(GEWorldPtr world, OSType elementID,
                        short hScroll, short vScroll);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Scrolling Element,

— hScroll and vScroll are the new horizontal and vertical components of this element's scroll position.

Alternatively, the application program can cause a Scrolling Element to scroll automatically at regular intervals by calling:

```
void AutoScrollGraphic(GEWorldPtr world, OSType id,
                            short interval, short autoHScroll,
                                        short autoVScroll);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the element,

— interval is the time between scrolls in milliseconds,

— autoHScroll and autoVScroll are the numbers of pixels to scroll the element horizontally and vertically after every interval.

To stop automatically scrolling an element, call AutoScrollGraphic() with autoHScroll and autoVScroll both equal to zero.

Frame Sequence Elements

Frame Sequence Elements are based on a sequence of pixel-map images. On the Macintosh, these are normally derived from PICT resources. Each of these images usually represents one frame in an animated sequence, though Frame Sequence Elements can also represent, for example, the "in" and "out" positions of a button. The application program creates a new Frame Sequence Element by calling:

```
GrafElPtr NewAnimatedGraphic(GEWorldPtr world, OSType id,
                             short plane, short resNum,
                             short mode, short xPos, short yPos,
                             short nFrames);
```

Where:

— the meanings of world, id, plane, resNum, xPos, and yPos are the same as for NewBasicPICT() above,

— mode is the graphic transfer mode used to draw the element (transparent in most cases, srcCopy for Frame Sequence Elements for which masks have been made),

— nFrames is the number of consecutively-numbered PICT resources, beginning with resNum, to be used in constructing the Frame Sequence Element.

The Graphic Elements system provides several procedures which can be used by the autochange procedure of a Frame Sequence Element (or the application program) to control the animation of the element.

The frame to be displayed can be set explicitly by calling:

```
void SetFrame(GEWorldPtr world, OSType elementID, short newFrame);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Frame Sequence Element,

— newFrame is the number of the frame to be displayed, which must be between 1 and the value of nFrames used when the element was created.

In most cases, Frame Sequence Elements are used to create animated graphics, and the frame will always change to the "next" frame in the sequence. Which frame is "next" depends on the kind of animation sequence being used and the current animation direction. Each Frame Sequence Element has a field element->seq which specifies the kind of animation sequence it uses. The Graphic Elements system supports the following kinds of animation sequences:

```
typedef enum {    singleframe=0,
                  reciprocating,
                  loop,
                  oneshotvanish,
                  oneshotstop,
                  oneshotloop} AnimSequence;
```

Where:

— singleframe means "no frame changes," and can be used to stop the animation of a Frame Sequence Element temporarily or permanently,

— reciprocating causes animation to begin at frame 1 and repeatedly sequence forward through the frames of the element to the last frame, then sequence backward through the frames to frame 1.

— loop causes animation to begin at frame 1 and repeatedly sequence forward through the frames of the element to the last frame, then start over with frame 1,

— oneshotvanish causes animation to begin at frame 1, sequence forward through the frames of the element to the last frame, then hide the element.

— **oneshotstop** causes animation to begin at frame1, sequence forward to the last frame, then stop, leaving the element visible.

— **onshotloop** causes animation to begin at frame 1, sequence forward to the last frame, loop to frame 1, then stop, leaving the element visible.

The application or autochange procedure can reverse the directions of these sequences by calling:

```
void SetAnimDirection(GEWorldPtr world, OSType elementID,
                                Boolean forward);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Frame Sequence Element,

— forward specifies whether the element's animation sequence is to run forwards or backwards. For example, if forward is **false**, an element with a loop animation sequence would start at the last frame, sequence backwards through its frames to frame 1, then start over at the last frame.

When the animation sequence and direction of a Frame Sequence Element have been set, the application or autochange procedure can advance the animation of that element to its next frame by calling:

```
void BumpFrame(GEWorldPtr world, OSType elementID);
void PtrBumpFrame(GEWorldPtr world, GrafElPtr element);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Frame Sequence Element, or (in the Ptr… version) element is a pointer to the element.

The PtrBumpFrame() variation is provided for situations in which a pointer to the element is already available, for example in the element's autochange procedure.

For “constant-motion” or “oneshot” Frame Sequence Elements, i.e. those which change frames automatically but do not otherwise require an autochange procedure, the Graphic Elements system provides the following procedure, which may be called during the initialization of such elements to set them in motion:

```
void AnimateGraphic(GEWorldPtr world, OSType elementID,
                        short interval, AnimSequence sequence);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Frame Sequence Element,

— interval is the time between frame changes in milliseconds,

— sequence is the desired kind of animation sequence, as described above.

To stop automatically animating a Frame Sequence Element, call AnimateGraphic() with an interval of zero.

For Frame Sequence Elements which are drawn using the built-in rendering procedure in transparent mode (or in srcCopy mode for elements for which "masks" have been made), the Graphic Elements system provides "source mirroring" so that the same source graphic can be used for screen images which are reversed right-for-left and/or bottom-for-top. The application program or autochange procedure sets mirroring for a Frame Sequence Element by calling:

```
void SetMirroring(GEWorldPtr world, OSType elementID,
                              Boolean mirrorH, Boolean mirrorV);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Frame Sequence Element,

— mirrorH and mirrorV control the horizontal and vertical reversal of the image: if mirrorH is true, the image will be reversed right-for-left; if mirrorV is true, it will be inverted bottom-for-top before being rendered in its GEWorld.

Mirroring a Frame Sequence Element does not change its location, as defined by its **animationRect**. However, mirroring will result in an *apparent* change in position if the element is not centered horizontally and vertically in this rectangle. And remember that mirroring affects all aspects of the element — for example, if one of the frames of a sequence includes a balloon with the word "POW", and the sequence is mirrored horizontally and vertically, the word will be upside-down and backwards.

Text Elements

Text Elements are Graphic Elements which show a single line of text in a font, size, style, and color specifed at creation. The application program creates a new Text Element and installs it in a GEWorld by calling:

```
GrafElPtr NewTextGraphic(GEWorldPtr world, OSType id, short plane,
                         short xPos, short yPos, short mode,
                         short fontNum, short txStyle, short size,
                         RGBColor color, StringPtr text);
```

Where:

— the meanings of world, id, plane, xPos, yPos, and mode have the same definitions as for NewBasicPICT() above,

— fontNum is the number of the font to use for this Text Element,

— txStyle is the text style for this element,

— size is the text size for this element,

— text is a pointer to a Pascal-style string containing the text for this element.

Because of the way in which text is drawn, the value passed for mode should almost always be srcOr for Text Elements.

The text displayed by an existing Text Element can be changed by calling:

```
void SetTextGraphicText(GEWorldPtr world, OSType elementID,
                                    StringPtr newText);
```

Where:

— world is the GEWorld containing the element,

— elementID is the unique four-character ID assigned to the Text Element,

— newText is a pointer to a Pascal-style string containing the new text for this element. Note that the Text Element *does not* make a copy of this string — the pointer passed as **newText** must remain valid as long as the element is in use.

Sensor Elements

The Graphic Elements system includes definitions for elements capable of acting as buttons, switches, and slider-type controls. The application program controls the position, visibility, and plane of a Sensor Element as it does for any other Graphic Element, by calling the display controller routines ShowElement(), MoveElement(), MoveElementTo(), and SetElementPlane(). The application program should also provide an action procedure for each Sensor Element it creates, and must call MouseDownInSensor() when the user presses the mouse button in the area

of a window containing a GEWorld in order to give the Sensor Elements in that world the opportunity to act.

To create a new button-type sensor, the application calls:

```
GrafElPtr NewButtonSensor(GEWorldPtr world, OSType id,
                                    short plane, short resNum,
                                    short xPos, short yPos);
```

Where:

— the meanings of world, id, plane, resNum, xPos, and yPos are the same as for NewBasicPICT() above.

Internally, a button-type sensor is treated as a Frame Sequence Element with two frames, one for the “off’ position and one for the “on” position of the button. The graphic for a standard button sensor should consist of two PICT resources, numbered resNum and resNum + 1 in the application program's resource file.

The tracking procedure for the standard button sensor follows Macintosh Human Interface Guidelines. While the user holds the mouse button down, the sensor goes “on” when the cursor is within its boundaries, ”off” when the cursor is outside. If the mouse button is released while the cursor is inside it, the sensor calls its action procedure. Note that, for button-type sensors, the sensorState parameter to this procedure is meaningless, since it will only be called if the button has been pressed. After the action procedure has returned, the button reverts to its “off” state.

A switch-type sensor differs from a button-type sensor only in that both the off-to-on and on-to-off transitions are meaningful. To create a new switch-type sensor, the application program calls:

```
GrafElPtr NewSwitchSensor(GEWorldPtr world, OSType id,
                                    short plane, short resNum,
                                    short xPos, short yPos);
```

Where all the parameters have the same meanings as for NewButtonSensor(). The tracking procedure for the standard switch sensor is similar to that of the standard button sensor. However, instead of going “off” when the cursor moves outside its boundaries, it reverts to its previous state. If the mouse button is released while the cursor is inside its boundaries, the switch-type sensor changes state and calls its action procedure with sensorState equal to its new state, either sensorOff or sensorOn.

The application program can explicitly set the state of a switch by calling:

```
pascal void SetSwitchState(GEWorldPtr world, OSType id,
                                                short newState);
```

Where:

- world is the GEWorld containing the Sensor Element,
- id is the unique four-character ID assigned to the Sensor Element,
- newState is sensorOff or sensorOn.

The Graphic Elements system includes both horizontal and vertical slider-type sensors. These sensors consist of two images — one for the background “scale” portion of the sensor, and one for the sliding portion. Their state is a number between 0 and 100, representing the position of the slider as a percentage of the distance from left to right (or bottom to top) of the “scale” graphic. The application program creates a new slider sensor by calling:

```
GrafElPtr NewSliderSensor(GEWorldPtr world, OSType id, short plane,
                                short resNum, short xPos, short yPos,
                          short sliderType, short handleResNum);
```

Where:

— the meanings of world, id, plane, xPos, and yPos are the same as for NewBasicPICT() above,

- resNum is the number of the PICT resource representing the background “scale” portion of the slider sensor,
- sliderType is hSlideSensor for a horizontal slider or vSlideSensor for a vertical slider,
- handleResNum is the number of the PICT resource containing the graphic for the sliding portion of the sensor.

The tracking procedure for a slider sensor follows the user's actions with the mouse as long as the mouse button is down. The Sensor Element's action procedure is called ”continuously,” every time the sensor's setting changes, with its sensorState parameter equal to the slider's new setting.

The application can explicitly set the position of the slider by calling:

```
pascal void SetSliderPercent(GEWorldPtr world, OSType id,
                                          short newSetting);
```

Where:

- world is the GEWorld containing the Sensor Element,
- id is the unique four-character ID assigned to the Sensor Element,
- newSetting is an integer between 0 and 100 (values < 0 or > 100 will be forced into this range).