Special Graphic Elements

As discussed in the section “Custom Graphic Elements”, any graphical entity can easily be converted to a Graphic Element. Because of the simplicity and completeness of the Graphic Elements paradigm, it also becomes possible to treat many things which are *not* precisely “onscreen graphics” as Graphic Elements. Examples of two such elements are included in the Graphic Elements system: a special-effects controller element, and a “grabber” element.

Special Effects Controllers

A Graphic Elements SFXController is a special element which:

— captures another Graphic Element (and all its dependents) in its present form by causing it to render itself into a new offscreen GWorld created for this purpose, then hiding the captured element;

— after a specified delay, calls a specified special-effects processor procedure a specified number of times at specified intervals;

— when the special effect is finished, deletes itself and frees all memory it has used, leaving the “captured” element either visible or invisible.

An SFXController is created and its effect is started by calling

```
GrafElPtr DoGESFX(GEWorldPtr world, OSType ctrlrID,
                                 GrafElPtr subjectElement,
                                 SFXProcessor fxProc,
                                 short nSteps, short delay,
                                 short stepTime, Boolean fxIn,
                                 Boolean forward);
```

Where:

— world is the GEWorld in which the new special effect is to take place.

— ctrlrID is the unique four-character name by which the application program will refer to this special effect controller.

— **subjectElement** is a pointer to the Graphic Element to which the special effect will be applied. This pointer can be obtained, for example, from **FindElementByID()**.

— **fxProc** is the processor procedure for this special effect. SFXProcessor procedures are discussed below.

- — **nSteps** is the number of steps in which the special effect is to be applied. Some, but not all, SFXProcessor procedures support variable numbers of steps.
- — **delay** is the number of milliseconds to wait before the first step of the effect.
- — **stepTime** is the number of milliseconds to wait between successive steps of the effect.
- — **fxIn** determines whether the original (captured) Graphic Element is to be left visible (**fxIn == true**) or invisible (**fxIn == false**) at the end of the special effect.
- — **forward** determines the direction of the special effect, in cases where "direction" is meaningful. For example, a horizontal wipe SFXController might wipe from left to right is **forward** is **true**, and from right to left if it is **false**.

Special effects are normally started from a Graphic Element's creation function, its collision function, or its autochange function. Note that, while a special effect is in progress, its subject element does not "exist" as far as the Graphic Elements system is concerned. The application program can determine when a special effect is finished by calling

```
Boolean SFXFinished(GEWorldPtr world, OSType ctrlrID);
```

Where

- — **world** is the GEWorld in which the special effect it taking place,
- — **ctrlrID** is the four-character ID assigned to the controller at creation.

SFXProcessor Procedures

A Graphic Elements special effects processor procedure has the prototype:

```
typedef pascal void (*SFXProcessor)(SFXCtrlrPtr controller);
```

Where

- — **controller** is a pointer to the special effects controller record for which processing is being performed.

The SFXProcessor uses fields of the SFXController record to determine what processing is needed on a given call:

— **controller->sfxSrc** is an offscreen GWorld containing the source graphic for the special effect.

— **controller->baseGraphic->graphWorld** is the offscreen GWorld which is the destination of the special effect.

— **controller->sfxData** can be used by the SFXProcessor to store a pointer to any additional data it needs to allocate.

— **controller->currentStep** is the step of the special effect for which the SFXProcessor is being called. See below for special meanings of **controller->currentStep**.

— **controller->nSteps** is the total number of steps for this special effect.

— **controller->forward** is **true** if the special effect should go "forward", **false** if it should go "backward".

The SFXProcessor is called two extra times during the life cycle of a special effect. When the SFXController is created, it is called with
controller->currentStep == 0. At this time, it may allocate memory (and assign a pointer to **controller->sfxData**) or do any other necessary preprocessing. It should not do any graphics processing — i.e., anything which affects **controller->baseGraphic->graphWorld** — at this time.

When the special effect is finished, the SFXProcessor is called with
controller->currentStep == -1. At this time, it should free any memory it has allocated, and perform any other necessary "teardown".

NOTE: There is no simple way for a special effects processor to recover from a failure to allocate memory. If it cannot allocate memory during its initialization step, it should store a **nil** pointer in **controller->sfxData**, and should be sure to check for a non-**nil** pointer in this field before using it.

NOTE 2: By their nature, digital special effects are expensive in both time and memory. A Graphic Element captured by a special effects controller temporarily takes up at least three times its normal memory, and can require more than double the amount of offscreen-to-offscreen copying. Be sure to allow for this when planning your special effects.

The "Grabber" Graphic Element

The "Grabber" is a Graphic Element which manipulates other Graphic Elements. It is used in software for interactively editing Graphic Elements Worlds and the elements they contain.

Essentially, the “Grabber” is a sensor-type element in the topmost plane of a GEWorld, with its sensitive rectangle set to the entire area of that world. When the user presses the mouse button, its tracking procedure searches for the topmost Graphic Element, in between a minimum and a maximum “plane” level, which contains the mouse point. If it finds such an element (and this element is not already selected), it drops its old selection and “grabs” the element, allowing the user to move it around in the GEWorld as long as the mouse button is down.

The application program creates a “Grabber” for a given GEWorld by calling

```
GrafElPtr MakeGrabber(GEWorldPtr world);
```

Where:

— **world** is a pointer to the Graphic Elements world for which the “Grabber” is being created.

A newly-created “Grabber” does nothing until it is activated. The application program should provide a means for the user to activate and deactivate the Grabber — while it is active, no other sensor in the GEWorld will function, since the user must be able to pick up and move these sensors just like any other Graphic Element. The application activates and deactivates the Grabber by calling

```
void ActivateGrabber(GEWorldPtr world, Boolean activate);
```

Where:

— **world** is the GEWorld containing the Grabber,

— **activate** is true to activate the Grabber or false to deactivate it.

The Grabber acts only on objects located between a minimum plane and a maximum plane. When it is created, the “lowest” plane is 2, and the ”highest’ plane is 32766. The application program can change this minimum and maximum at any time by calling

```
void SetGrabberDepth(GEWorldPtr world, short newMinPlane,
                                          short newMaxPlane);
```

Where

— **world** is the GEWorld containing the Grabber,

— **newMinPlane** is the new “lowest” plane on which the Grabber will grab Graphic Elements,

— **newMaxPlane** is the new “highest” plane on which the Grabber will select elements.

Set **newMinPlane** and **newMaxPlane** to the same value to cause the Grabber to select only the Graphic Elements on a single plane. The minimum plane should always be “higher” than the plane of the background, so that the Grabber will not pick up the background graphic, and ”lower” that the highest possible plane (32767), so that it won't try to grab itself.

The application program can find out whether a Grabber is active by calling

```
Boolean GrabberActive(GEWorldPtr world);
```

Where

— **world** is the GEWorld containing the Grabber,

The application program can retrieve the Grabber's current selection at any time by calling

```
    GrafElPtr  CurrentGrabberSelection(GEWorldPtr world);
```

Where

— **world** is the GEWorld containing the Grabber,

When the Grabber has no current selection, this function will return **nil**. The application program must be sure to take this possibility into account.