The NewTek Video Toaster Developer's Handbook

Revision 10/26/95
Compiled and Published by Daniel Wolf
for the
NewTek Developer Support Program

The contents are separately paginated stand-alone documents.

Disclaimer

NewTek, Inc. provides these materials at its sole discretion to aid developers of third party products which enhance and extend the capabilities of NewTek's own products. NewTek, Inc. maintains a policy of continuous improvement of its own products and therefore:

All documentation and specifications contained herein are subject to change without notice.

NewTek, Inc. provides these materials with no warranty or guarantee whatsoever.

To arrange for specific ongoing or updated support, please contact NewTek, Inc.

An NTSC Primer
by D. Wolf
7/4/95

NTSC = National Television Standards Committee PAL = Phase Alternating Line

NTSC color video adds color information to what was originally a monochrome video signal. RCA proposed this method of encoding color at a resolution substantially lower than luminance and adding that information to the luminance signal via quadrature phase modulation (4 possible values of phase) of a specific frequency already within the luminance bandwidth.

The frequency of 3.57 mHz was chosen for the color subcarrier, which gets phase modulated with color information.

Problems can arise. When a NTSC video signal happens to have luminance information at the same frequency as is used for encoding color, false colors can be induced where none exist (seen when closely spaced vertical stripes of luminance are broadcast - like a performer wearing a striped tie or a football referee's shirt). Likewise, when the color frequency isn't (and almost never is) perfectly extracted from the overall signal, information from the color process 'leaks' into the luminance signal and appears as dot crawl. The following additional information about NTSC (see the phase reversal discussion on successive fields and lines below) should clarify why the dot crawl looks like it does.

Here are some NTSC fundamentals.

***** Definition of Luminance:**

Y = 92.5 IRE 'Gray' (White)

(1) Y = .30*R + .59*G + .11*B

Color might have been expressed as R-Y and B-Y, but they're a bit different for reasons of transmission and encoding efficiency. Color signals are expressed as I and Q and they are related to R-Y and B-Y as follows.

***** Definition of Color I and Q: Quadrature Modulation**

(2) Y-Axis is (R-Y) scaled by 1/1.14 = .877
(3) X-Axis is (B-Y) scaled by 1/2.08 = .493

The Q-axis is 1 radian (57 degrees) clockwise from (R-Y)
The I-axis is 1 radian (57 degrees) clockwise from -(B-Y)

(4) (R-Y) = 1*R - .30*R - .59*G - .11*B = .70*R - .59*G - .11*B
(5) (B-Y) = 1*B - .30*R - .59*G - .11*B = -.30*R - .59*G +.89*B

(6) cos(pi/2-1) = cos(33 deg) = .84147
(7) sin(pi/2-1) = sin(33 deg) = .54030

Successive pixels are samples of Y+I, Y+Q, Y-I, Y-Q as the color sampling rotates clockwise. So, by applying equations (1) through (7), we obtain the equations for I and Q.

(1) Y = .30*R + .59*G + .11*B
(8) I = cos(pi/2-1)*.877*(R-Y) - sin(pi/2-1)*.493*(B-Y) = .60*R - .28*G - .32*B
(9) Q = sin(pi/2-1)*.877*(R-Y) + cos(pi/2-1)*.493*(B-Y) = .21*R - .52*G + .31*B

Color phase on line 1 of Field 1 is 0 degrees: Y+I Y+Q Y-I Y-Q . . . (1st pixel, 2nd pixel, etc.)
Color phase on line 1 of Field 2 is 180 degrees: Y-I Y-Q Y+I Y+Q . . .
Color phase on line 1 of Field 3 is 180 degrees: Y-I Y-Q Y+I Y+Q . . .
Color phase on line 1 of Field 4 is 0 degrees: Y+I Y+Q Y-I Y-Q . . .

Color signals are encoded as quadrature modulation of a 3.57 mHz phase added to the luminance (monochrome) waveform. One complete cycle of the color subcarrier (3.57 mHz) has four possible phase positions, either I, Q, -I, or -Q.

Since color is 'added' to the luminance, and D2 samples the NTSC waveform at 14.28 mHz (4x the 3.57 mHz frequency), each successive sample catches a single color phase position. Successive D2 samples are Y+I, Y+Q, etc. The specific ordering depends on which field is being sampled; Fields 1 and 3 are in order of Y+I, Y+Q, Y-I, Y-Q (0 degrees color subcarrier reference phase) and Fields 2 and 4 are in order of Y-I, Y-Q, Y+I, Y+Q (180 degrees color subcarrier reference phase). NTSC essentially dithers the I and Q signals 1 pixel apart on successive lines of two successive fields. On two successive fields (a NTSC frame), a specific line of the field has opposite phases. On two successive frames, the phase reverses. NTSC thus dithers I and Q in both space (line by line in a frame) and time (even and odd lines of a frame have reversed phases in successive frames).

This should explain the why's and wherefore's of Toaster QUAD usage in digital warps and also why the Flyer's D2 'live' video comes in 2-frame (4-field) temporal chunks.

Here are the basic YIQ and RGB equations [see equations (1), (8), and (9), above]:

***** YIQ from RGB Coefficients**

yr = .30
yg = .59
yb = .11

ir = .60
ig = -.28
ib = -.32

qr = .21
qg = -.52
qb = .31

***** YIQ from RGB**

[YIQ] = [RGB] [yr ir qr
yg ig qg
yb ib qb]

Y = yr*R + yg*G + yb*B
I = ir*R + ig*G + ib*B
Q = qr*R + qg*G + qb*B

Y = .30R + .59G + .11B
I = .60R - .28G - .32B
Q = .21R - .52G + .31B

***** RGB from YIQ Coefficients**

ry = 1.000
ri = .9483
rq = .6240

gy = 1.000
gi = -.2761
gq = -.3298

by = 1.000
bi = -1.1055
bq = 1.7299

***** RGB from YIQ**

[RGB] = [YIQ] [ry gy by
ri gi bi
rq gq bq]

R = ry*Y + ri*I +rq*Q
G = gy*Y + gi*I +gq*Q
B = by*Y + bi*I +bq*Q

R = 1.00*Y + .9483*I + .6240*Q
G = 1.00*Y - .2761*I - .6398*Q
B = 1.00*Y - 1.1055*I + 1.7299*Q

References:

Code of Federal Regulations 47 - Part 73.682
Video Cameras: Theory and Servicing by Gerald R. McGinty (Howard W. Sams & Co, Inc. Books, 1984)

CALCYIQ.C

```
/************************************\
*                                    *
* Converts a pixel quad into YIQ values *
*                                    *
\************************************/

#include "exec/types.h"
#include "exec/memory.h"
#include <proto/dos.h>
#include "libraries/dos.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>


main(argc,argv)
int  argc;
char  **argv;
{
      int   p1,p2,p3,p4,Y,I,Q;

      if (argc!=5) {
            printf("Usage: %ls <p> <p> <p> <p>\n",argv[0]);
            exit(100);
      }

      stch_i(argv[1],&p1);
      stch_i(argv[2],&p2);
      stch_i(argv[3],&p3);
      stch_i(argv[4],&p4);

      Y = (p1 + p2 + p3 + p4)/4;
      I = ((Y - p1) + (p3 - Y))/2;
      Q = ((Y - p2) + (p4 - Y))/2;

      printf("Y=%ld, I=%ld, Q=%ld\n",Y,I,Q);
}
```

FXDox by Daniel Wolf ©1994 by NewTek
Started - November 15, 1994
Version - November 16, 1994
Version - November 28, 1994
Formatted to PageStream - December 1, 1994

NewTek Video Toaster FX Theory

The Video Toaster Pixel Switching and Fading Hardware

The basis of Toaster FX capabilities we'll discuss resides in three video processing circuits on the Toaster's digital video board. Please refer to Figure 1, the Toaster Digital Video Block Diagram. Two of these circuits (labelled AMUX and BMUX) are analog video source selectors (multiplexers) for the third circuit, a two-source video mixer (labelled A/B FADER). Because the output of the A/B Fader is the Toaster's Main Video output, controlling the Fader is the key to Toaster video processing. In this discussion, we'll trace the Toaster's Main output backwards and see how Toaster circuitry can direct different video sources (Toaster inputs, Digital Stills, Amiga graphics, etc.) out to Main. The heart of the Toaster's video processing capabilities is the first stage 'back' from the Main output line, the high speed digital fader (far right on the block diagram).

The Digitally-controlled Video Mixer/Fader

The Fader is always fed two analog video signals, from the AMUX and BMUX along with a 8-bit binary control signal. The Fader can mix the A and B analog video inputs in 256 levels (dictated by the 8-bit binary control signal) at very high speed. The fader can reset mixing level of A and B inputs (i.e. it can respond to a change of value on its 8-bit control signal lines) every 70 nanoseconds (ns). That is fast enough to change A/B video mixing levels 'on the fly', pixel by pixel (about 768 pixels on each line of video - roughly equivalent to Amiga HIRES video). The Fader can select either of its inputs or a mixture of them and route that to the Main output on a pixel by pixel basis.

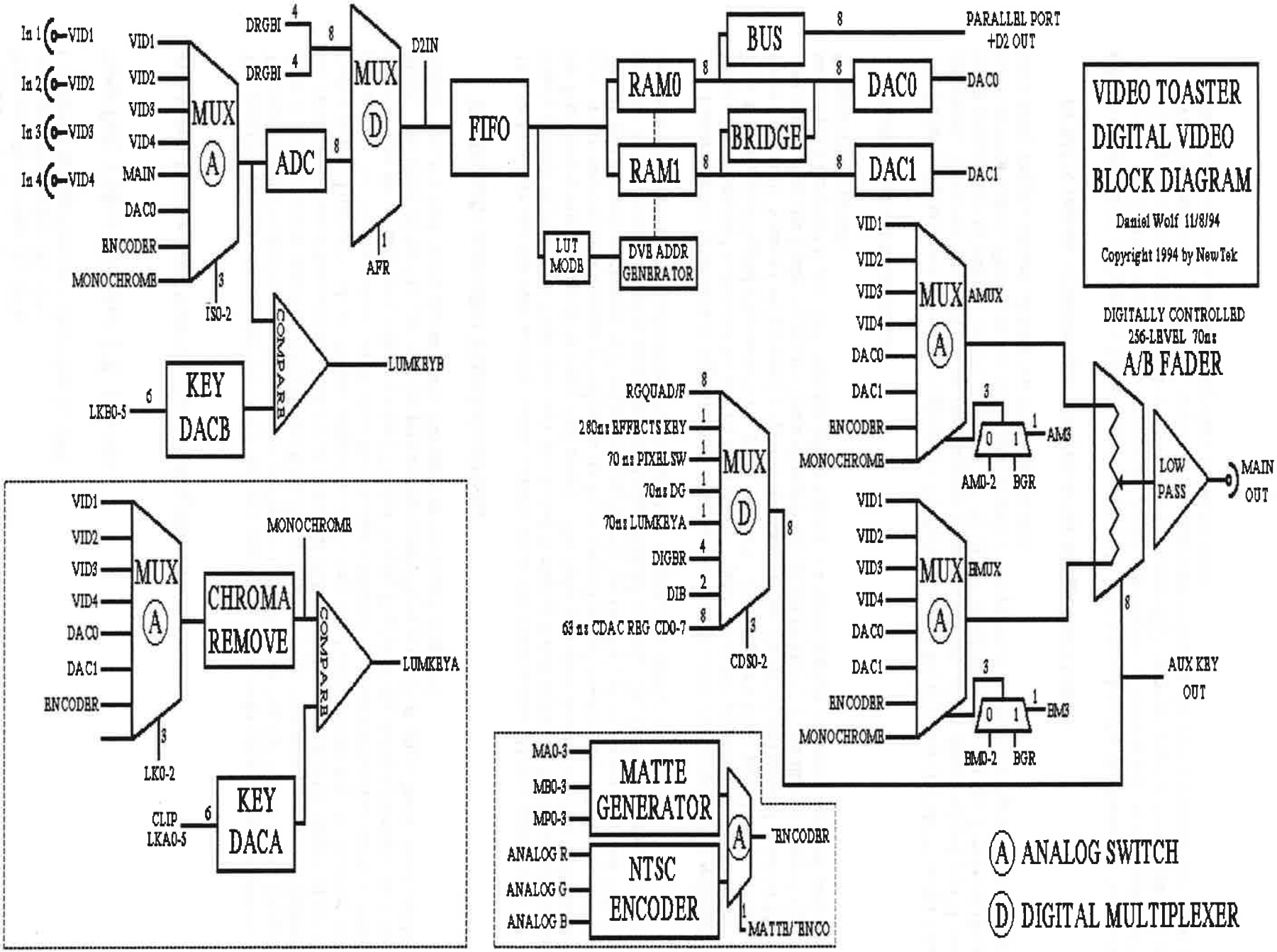
The Fader's 8-bit (256-level) digital control signal sets its Source1/Source2 mix combination. Source1 and Source2 video can be any of the Toaster's possible video signals, e.g. Video Input 1, 2, 3, or 4, or Toaster Digital Channels DV1, DV2, or DV3. If the Fader's 8-bit digital level is 0 then the Fader selects its Source1 video input to go out to Main. If the Fader's level control is set to 255 (its maximum) then the Fader routes its Source2 video signal out to Main. If the digital control signal is somewhere between 0 and 255 then the Main output is a combination of partly Source1 and partly Source2 video signals.

Fader Control Multiplexer

The Fader's digital control values originate in the Fader Control Multiplexer (FCMUX on the block diagram). Examine the FCMUX on the diagram (near the middle of the diagram) and you'll see it has 8 inputs: any of those 8 inputs to the FCMUX can be selected to govern the 256-level (8-bit) fader control value. The inputs to the FCMUX which can provide the Fader's digital control signal have fairly cryptic names, so here's a brief explanation.

FCMUX Inputs - Controllable Directly or by Amiga Pixel Colors

63mus CDAC REG - These are 8 bits which can directly feed through to the the Fader's 8-bit control lines. The CDAC can actually change the Fader's 8-bitcontrol lines at a time resolution of 63 microseconds (one change of this setting per line since 62.5 microseconds is the duration of one NTSC video line). The CDAC register is usually controlled by the user's input to the T-bar on the Switcher screen, though, only once per video field.


VIDEO TOASTER
DIGITAL VIDEO
BLOCK DIAGRAM
Daniel Wolf 11/8/94
Copyright 1994 by NewTek
In 1
In 2
In 3
In 4
VID1
VID2
VID3
VID4
MAIN
DAC0
ENCODER
MONOCHROME
MUX
A
ISO-2
ADC
DRGBI
DRGBI
MUX
D
AFR
D2IN
FIFO
LUT MODE
DVE ADDR GENERATOR
RAM0
RAM1
BUS
BRIDGE
DAC0
DAC1
PARALLEL PORT
+D2 OUT
COMPARB
LUMKEYB
KEY DACB
LKB0-5
DIGITALLY CONTROLLED
256-LEVEL 70ns
A/B FADER
AMUX
HMUX
AMS
HMS
AM0-2
HM0-2
BGR
LOW PASS
MAIN OUT
AUX KEY OUT
RGQUAD/F
280ns EFFECTS KEY
70 ns PIXELSW
70ns DG
70ns LUMKEYA
DIGBR
DIB
63 ns CDAC REG CD0-7
CDS0-2
CHROMA REMOVE
LUMKEYA
LK0-2
CLIP
LKA0-5
KEY DACA
MA0-3
MB0-3
MP0-3
ANALOG R
ANALOG G
ANALOG B
MATTE GENERATOR
NTSC ENCODER
ENCODER
MATTE/ENCO
A ANALOG SWITCH
D DIGITAL MULTIPLEXER

RGQUAD - These are 8 bits derived from the high bits of RED and GREEN Amiga color values of four neighboring pixels (a 'QUAD' of pixels) on the Amiga's display (see DIGBR, below).

QUAD/F - This is a single bit which is 1 if a QUAD is within a certain legal range of color values or 0 if the QUAD of pixels is outside that range.

70ns PIXEL SWITCH - This is a single bit which is 1 when the Amiga's genlock system detects that the color on the Amiga's display is a 'genlock transparent' color. On Amiga 2000, the only 'genlock transparent' color possibility is the Amiga display's background color (the zero color in the Amiga's display palette). On the Amiga 4000, the 'genlock transparent' color can be any single color in the Amiga's display palette.

70ns LUMKEYA - This is a single bit which is 0 if the user-selected Toaster 'Clip Level' is below the threshold for luminance keying transparency. This bit is a 1 when Toaster luminance keying is actually active for a particular pixel.

70ns DG - This is a single bit derived from the high bit of the GREEN Amiga color value of a single Amiga pixel (70ns Digital Green bit). It is a specific data line on the Amiga's video slot (see DIGBR, below).

DIB - This is a two bit signal derived from the Amiga's digital video output. These are the two most significant bits of the DIBGR signal described next.

DIBGR - This is a four bit signal which derives from the Amiga's digital RGBI lines (present on the 23-pin RGB video connector and - most importantly for the Toaster - on the video slot!). Most Amiga users view their Amiga displays as 'analog RGB' using a typical 1084 monitor. The 'analog RGB' on an A2000 provides up to 4096 different color values (Amiga ECS and pre-ECS 12-bit color). The 'analog RGB' on an A4000 provides up to 16 million different colors. The digital RGB signals, on the other hand, are quite limited in comparison and that's why most users never bother to view them. On an A2000, there are only 16 possible colors using the Amiga's digital RGB signals - 1 bit each of Red, Green, Blue, and 'Intensity' (8 colors, each with bright and dim levels). The purpose for which the Digital RGB signals were originally included in the Amiga design was to make the Amiga compatible with IBM/clone CGA-style graphics monitors. That type of monitor did indeed have only Digital RGB color inputs and displayed only 8 different colors (combinations of the R, G, and B digital bits) at two different intensity levels (the I bit). The individual DIGBR (Digital Intensity, Green, Blue, and Red) bits come from the high bits of an Amiga palette register's RED, GREEN, and BLUE levels (which themselves are 4 bits each on an Amiga 2000 and are 8 bits each on an Amiga 4000) as each Amiga pixel goes by. These four bits of color information are the Toaster's sole digital video control inputs from the Amiga! The reason they are key to the Toaster's capabilities is that the Amiga updates these four bits of Amiga color information (meager as they are) at an extremely rapid rate - namely every 70 nanoseconds! They provide the Toaster with digital information about the Amiga's display at precisely the correct rate to permit all the Toaster's digital video wizardry. The DIGBR is four bits of digital information which occurs every 70ns.

Note that on an Amiga 2000 there really are four individual bits named R, G, B, and I on the video slot data lines, but on an A4000 there are more digital RGB data lines but no I data. The Video Toaster works similarly on both the A2000 and A4000 and the names of control signals on the A4000 Toaster have been kept from the older A2000 version, but the actual Amiga RGB digital data lines used by Toaster 4000 circuitry may differ from those used by Toaster 2000 circuitry. For the purposes of this presentation there is no need to distinguish between them.

The Fader's sources of level controls represent a wide variety of possibilities! All but one source of the Fader level controls are derived from Amiga graphics in one way or another. The CDAC is the only exception. Some of the Fader control sources are 8 bit values (like RGQUAD and CDAC) which can drive the Fader's full range of values. Other Fader control sources (like DIB, which is

only 2 bits of data) provide less 'resolution' of Fader source control. For example, the DIB is only two bits of data and can only generate four levels of Fader action. The LUMKEYA is only a single bit and therefore it can only tell the Fader to go 100% to its A source or 100% to its B source. The number of Fader levels can't exceed the possibilities provided by the FCMUX control source which is currently selected. Suffice it to say, for now, that the Fader control sources mostly provide pixel-by-pixel Fader level capability, but the selection of the Fader control source can only happen once for each line of Toaster video. The Toaster's software manipulation of Fader control source (selecting which source to the FCMUX is actually in use on any particular line of Toaster video) is discussed later.

Since there are 8 possible Fader control sources, it takes 3 bits of information (a 3-bit number, from 0 to 7) to select one of those eight possibilities for controlling the Fader's levels. That numerical selection value for the FCMUX comes from 3 digital bit lines called CDS0, CDS1, and CDS2 (CDS0-2 on the block diagram).

In summary, the digitally controlled 70ns Fader is one of the key Toaster video processing elements. The Fader can smoothly mix two sources of video on-the-fly. Fader video mixing levels are themselves controlled by any of 8 different methods - mostly derived from actual Amiga graphics displays.

Fader Video Input Sources - AMUX and BMUX

The Fader always has two analog video signal inputs, which it can smoothly and rapidly switch or combine to create the Toaster's Main video output. Both of these two video signal sources for the Fader comes from a pair of switchable selectors (multiplexers or analog switches). These are appropriately named the AMUX and BMUX (A multiplexer and B multiplexer). Like the FCMUX, the AMUX and BMUX each have 8 different input sources which can be routed through to the Fader, but the AMUX and BMUX input sources are single-line analog video signals, unlike the FCMUX input sources which are multiple digital control bit lines. Here's a list of the video signal sources which can be routed through the A and B multiplexers.

VID1 - 'live' NTSC analog video input 1 to the Toaster

VID2 - 'live' NTSC analog video input 2 to the Toaster

VID3 - 'live' NTSC analog video input 3 to the Toaster

VID4 - 'live' NTSC analog video input 4 to the Toaster

DAC0 - video from the Toaster's #1 Digital Video Bank (DV1), after being converted into NTSC analog video by the Toaster's Digital to Analog converter (DAC) #0

DAC1 - video from the Toaster's #2 Digital Video Bank (DV2), after being converted into NTSC analog video by the Toaster's Digital to Analog converter (DAC) #1

ENCODER - Amiga RGB graphics converted by the Toaster into NTSC analog video. See the lower center portion of the block diagram and note the dotted-line region that shows the Toaster's circuitry for creating either a selectable background color (MATTE GENERATOR) or Amiga RGB graphics converted to NTSC video by the NTSC ENCODER.

MONOCHROME - A NTSC analog gray-scale video signal generated from any o the above sources by a color-removal circuit within the Toaster. See the lower left part of the block diagram and note the dotted-line region that the Toaster circuits which generates monochrome and luminance keying signals from any of the above 7 possible analog video sources.

AMUX and BMUX Input Selection - Controllable by Amiga Pixels

Each of the AMUX and BMUX can pass through any one of these 8 possible video signals to the Fader. Like the FCMUX, a 3-bit digital quantity selects which of the 8 possible video signals goes through each of the A and B multiplexers. Note the little AMUX and BMUX selector circuits on the block diagram (just below and to the right of the AMUX and BMUX diagrams). Unlike the FCMUX, which simply has CDS0-2 direct selector controls, the AMUX and BMUX have a more complicated means of selecting which input gets passed through to the output (to the Fader). Again, 3 bits are required to select one of the 8 AMUX or BMUX inputs, but AMUX and BMUX have another level of control. These little control multiplexers allow multiple ways in which AMUX and BMUX inputs can be selected to pass through to the Fader. The most important thing to notice here is that the little AMUX and BMUX control multiplexers have a special capability - they can select an input on AMUX or BMUX more or less directly (signals labelled AM0-2 and BM0-2, like the FCMUX's CDS0-2 selector lines) but (IMPORTANT) the AMUX and BMUX can also let their input selection be controlled by the Amiga's B, G, and R bits! See the discussion on DIGBR and Amiga digital color bits, above. Amiga pixel colors can control which video signal passes through the AMUX or BMUX out to the Fader. Because the Amiga supplies new B, G, and R bits on EVERY PIXEL, the decision about which video source to send to the Fader CAN CHANGE ON EVERY PIXEL!

Note that there are some limitations on combining effects modes because some of the same control bits used for AMUX and BMUX source selection are also used to manipulate the Fader. On the Amiga 2000 it is not practical to combine AMUX/BMUX and Fader control at all. On the Amiga 4000 the DIB control bits can indeed handle Fader control while other bits simultaneously interact with AMUX/BMUX source selection - hence the wider range of more sophisticated Toaster effects possible on the Amiga 4000.

Summary

The heart of the Video Toaster's Digital Video Effects is the combination of two 8-source multiplexers (AMUX and BMUX), each of which feeds a video signal to a fast, controllable Fader. All three of these circuits can be controlled on-the-fly, pixel-by-pixel. The sources of video being routed to the Fader's inputs can change among 8 possibilities AND the Fader itself can arbitrarily mix the two active sources - all at a temporal resolution of 70 nanoseconds. The Toaster's Main video output is a variable mixture of any two of eight possible video signals - both the mixture level and the selection of which two of the eight source signals can vary pixel by pixel. The mixture level AND the two sources can not only vary on a pixel-by-pixel basis, but the Amiga pixels' color values can provide all the control of both the video source selections and fader level.

When you combine this control flow design of the Video Toaster with the capability of the Amiga to play hi-resolution animations (which change the pixel colors in an orderly way, video field by video field, up to 60 times per second) you can see how the Toaster/Amiga combination leads to a real-time ('animated') digital video processing and digital video effects system.

Designing FX

The Toaster is capable of dozens of types of digital video effects, generated by the interaction between Amiga animations and the Toaster video signal flow control capabilities discusses above. We'll limit this discussion to four of these myriad possibilities. First let's think through some of the options.

Terminology

Latch - selecting video sources to the Fader on the AMUX and BMUX

Alpha - a computer graphics term to describe variable transparency between two images, implies Fader manipulation of variable levels of two video source images on the Toaster's output

Key - a video term implying overlay of one video image over another with the mixing level (which portions of each image appear on the 'keyed' combination) controlled by brightness (luminance keying) or color (chroma keying) values in the two source images. 'Keying' is the video jargon analog of 'alpha'.

DIB - Digital Intensity & Blue, two Amiga digital video bits used to control Toaster 'keying' operations

Encoder - Video derived from electronically converting Amiga RGB to NTSC

Matte - solid background and border color (generated within the Toaster)

CurrentMain - found frequently in the NewTek effects include files, implies 'whatever is currently on the Main video buss' (not necessarily any particular video source, but 'whatever') - in almost all effects this is the Source1 (starting source, the source FROM which the Toaster is switching). In this text description, I've used the term 'Source1' to mean the same thing.

CurrentOverlay - found frequently in the NewTek effects include files, implies 'whatever is currently on the Overlay video buss' (again, not necessarily any particular source, but 'whatever') - in almost all effects this is the Source2 (ending source, video TO which the Toaster is switching). In this text description, I've used the term 'Source2' to mean the same thing.

CrUD - Crouton User Data, color and video source control bit data portion of all Toaster FX files

Single Image Color Cycling Source/Encoder/Fade FX (NEON BANDS)

In this effect, a single Amiga 256-color hi-res interlaced image is the source of all the action. The Toaster smoothly changes its Main output from one video source to another as colors cycling through the image successively induce different areas of one source to become more and more transparent to the other video source. As part of the action we also see portions of the actual Amiga image as its colors cycle. Obviously, since we see some Amiga graphics, the Encoder (discussed above) is sometimes passed through AMUX or BMUX to the Fader. We also see a combination of the user-selected Main and Preview sources with the 'encoded' Amiga graphics at varying transparency (each second source increasingly shows through the increasingly 'transparent' Encoder source). We might guess that at any one pixel on the Main Toaster output's display, we see the Fader mixing between either the user-selected Source1 (Vid1, Vid2, Framestore, etc.) or Source2 and the Encoder. We might imagine that AMUX is being switched on the fly so that either of two Toaster video sources is fed to the A side of the Fader and Encoder (Amiga graphics) is being fed continuously to the B side of the Fader through the BMUX. Then the Fader is being manipulated to output varying combinations of what comes through the two MUX's.

That is indeed what is happening in this effect, and the colors on the Amiga graphics are being manipulated so that their Amiga digital color bits control the Toaster's video flows for this combination of 3 sources of imagery to generate a smooth transition from Source1 to Source2 on the Toaster's Main video output. The palette colors through which the Amiga image cycles have their low-order bits set to manipulate the Toaster's video signal flow in just the right way. Here's how:

One of the color palette 'bits' in each color is used to force AMUX to select between Source1 and Source2. (Yes, Virginia, it does take 3 bits to select one of the 8 AMUX inputs to pass through to the A side of the Fader, but the Toaster will take care of this automatically for our purposes. Since we only wish to choose between the two sources selected on the Switcher Interface we can think of this as a single bit while designing the effect. The Switcher software does indeed manipulate all three

'latch' bits in the palette colors behind the scenes.) Encoder is 'permanently' selected on BMUX (that is set up by the Toaster in a way which doesn't require any on-the-fly input from Amiga image color bits since it is not changing - more on that later but 'hint' - remember the little mini-muxes which control how AMUX and BMUX input selection takes place?), and two bits of each color are used to vary the Fader's level of mixing between its A and B inputs. Those two bits of Fader control let the Fader show four levels of mixing between its A and B sides. Altogether, 3 different sources of video are being mixed and the actual palette values assure that the mixing goes in an orderly way as the different Amiga RGB color values cycle through the Amiga image's palette. Obviously the initial color palette positions in the Amiga image have color values which assure that ONLY Source1 is solidly on the Fader's output and at the end of the color cycling the same image's palette has color values which assure that ONLY Source2 is solidly on the Fader's output. What happens in between is that a 'longer' palette of colors, is 'slid' through the image's palette (starting with all Amiga colors having Source1 control bits and ending with all Amiga colors having Source2 control bits) and the intermediate colors in the long palette have various bits sets to manipulate the AMUX and the Fader.

This type of color cycling fade transition can only be used with an Amiga A4000 since it uses a 256-color palette (really only 240 colors are actually used out of the 256-color palette, others are reserved but more on that later). This is an 'AA-only' effect. An Amiga A2000 can't use this particular kind of effect with so many smoothly changing colors because an A2000 is limited to hi-resolution images of only 16 colors. An variation of this effect could be designed for an A2000 system but it would have to conform to the limitations of the A2000's color versus resolution tradeoffs. We'll come back to this effect later and show how to use its theory of operation to make similar ones of your own.

4-Color Animated Source/Source Switch FX (CAMERA IRIS)

This type of effect is perhaps the simplest of the animated effects. A palette of only 4 colors is used on the Amiga animation and only 3 of them play a role in the video switching. In the CAMERA IRIS effect we first see Source1 solidly on screen as a nearly solid black animation of the camera iris slowly closes down around it and covers it up. As the iris opens again the Source2 video appears at the screen's center and expands from the center of the screen to reveal all of Source2. In this effect the Fader is not moved at all - only the BMUX is rapidly switched between video sources under control of the 3 of the animation's palette colors.

For the first half of the animation, AMUX alternately selects Source1 and the Toaster's Matte background color (set to black). These two sources are selected on AMUX according to two of the colors in the animation. In this first half of the animation's duration only 2 Amiga colors are in use in the animation. One of them is a color that lets Source1 show in the Iris' center as it closes down, the other selects the Toaster's Matte background color on AMUX.

During the second half of the animation one of the Amiga's colors is changed and that lets the AMUX now select Source2 or Encoder Matte as its input. This portion of the animation has the alternate color (the one which lets Source2 get selected on the AMUX) occupy the center of the Iris' opening.

I've left out a tiny bit of this animation's complexity since during the first half, there are really 3 sources of video alternately applied to AMUX by various pixels of the Amiga's animation. There are some thin lines in the animation which ALWAYS show the Source2 video, throughout the whole animation, i.e. the color of the animation which lets Source2 become selected on the AMUX also appears in a tiny portion of the animation - during both halves of the animation. During the second half of the animation, only 2 colors are used as described above - causing switching of AMUX between only the Source2 video and the Toaster's Matte generator (set to black).

Remember, during this entire animation, the Fader simply sits with 100% of its output coming from the AMUX and the AMUX does ALL the work by being controlled by 3 different Amiga animation palette colors. We'll also come back to this effect later to show how its theory of operation can be exploited for effects of your own creation.

4-Color Kiki Source/Matte/Source Wipe FX (BEAR)

This type of effect is produced in a way identical to the CAMERA IRIS effect, only the animation moves across the screen instead of playing statically centered on the Amiga screen.

Multi-Color Smooth-edged Animated Wipe FX (GIRAFFE)

This type of effect has features in common with both the color cycling static image (NEON) and the simple 4-color moving or static animated wipe (CAMERA IRIS) described above. These wipes use a full color palette (actually, 240 out of the 256-color full palette) and an Amiga animation, but these animations have plenty of color and visual appeal beyond the simple shadow-like animation of CAMERA IRIS or BEAR. This type of effect also makes use of varying degrees of transparency between the A and B sources applied to AMUX and BMUX by swinging the Fader smoothly between them. Like the NEON effect, the AMUX is switched pixel-by-pixel between two user-selected video sources and the Amiga images are applied as the only source to the BMUX via its Encoder input. The use of the Fader to show mixtures of the AMUX and BMUX signals lets us have smooth edges around the borders of the animated object moving across the screen.

The GIRAFFE effect will serve as an example of this type of effect but there are many which are created in a similar way, including a number of the wipes which have other objects in place of the giraffe. Some of those related effects may use a scrolling image which is larger than the display area instead of a regular Amiga animation (simpler when the image isn't really changing like the giraffe does) but the idea is the same.

Introducing a variable transparency or mixing (Fader control) along with the animation smoothly anti-aliases edges on the animation's Amiga (Encoder signal) graphics where they adjoin either the Source1 or Source2 video. This helps minimize jaggies along the edges of the Amiga animation and NTSC video artifacting which can occur along sharp-edged color transitions.

This type of effect uses a non-cycling palette but the construction of the palette is peculiar because of the method used to build the animation. The animation's color palette has to include colors whose bits force Fader levels and AMUX source variations like the NEON cycling effect's varying palettes do.

The construction of this type of Amiga animation is a 3-step affair. Three different animations are actually merged into one (using a tool called BitPlay we'll meet shortly). One of the animations is a simple two-color (a single Amiga bitplane) Amiga animation which is used in the end result to specify the bit-control of the AMUX (selection of Source1 or Source2 (remember that Source1 is just 'whatever' is on the Main buss, Source2 is 'whatever' is on the Overlay buss - Source1 is the 'CurrentMain' and Source2 is the 'CurrentOverlay' - Source1 is the 'from' and Source2 is the 'to') video inputs to the A side of the Fader). Another animation is a four-color (two bitplanes) Amiga animation and it is used for the anti-aliasing variable-transparency edges of the Amiga graphics. The third animation is a 32-color (5 bitplanes) Amiga animation, which becomes the Encoder video signal on the BMUX's side of the Fader. The 32-color animation is the original full-color GIRAFFE animation. The original 32-color animation is made intentionally NOT to use the top two Amiga colors (does NOT use Colors #30 and #31!). When BitPlay merges the three animations together, the 32-color Amiga animation's colors are spread out into a palette of 256 colors (again, only 240 are actually used with 16 reserved for specific Toaster internal use). Each of the Amiga animation's original 32 colors is spread into 8 sub-shades by the introduction of the 3 other bitplanes from the other animations (1 bitplane for AMUX selection and 2 for Fader control). The top two colors (#'s 30 and 31) were not used in the original animation so that they become colors #240-255 when the anim is manipulated by BitPlay and those colors need to be reserved for the Toaster's own use. Avoiding Colors #30 and 31 when making the original anim just assures that the top 16 colors of the 256-color anim created by BitPlay are left unused. The 8 subshades within a color group are nearly indistinguishable visually, only a few low order color bits vary among them. They can be distinguished when you examine frames of the composited animation after merging with BitPlay.

Now we need to exercise a bit of mental gymnastics to imagine how a single the effect of either a 1 or 0 in a pixel of one bitplane of an 8 bitplane Amiga graphic becomes reflected in an Amiga 256-color palette. Let's keep it simple and imagine that the bitplane in question is the low order bitplane. If a 1-bit is present in a pixel of that bitplane then it must be true that one of the odd-numbered colors in the palette will be used for that pixel. Remember that when an Amiga graphic is made of 8 bitplanes (only possible on an AGA Amiga, of course), the Amiga takes the 8 bits of a pixel which originate in different bitplanes and uses that 8-bit value to select a color for that pixel from a table of 256 colors. Now you can see why the presence of a 1-bit in a pixel's low order bitplane will mean that some odd-numbered color palette position is selected to color that pixel. If a 0-bit occurs in a pixel's low order bitplane then the color selected will be one of the even-numbered palette entries. The overall effect of using a single bitplane of an Amiga animation to specify the Toaster's AMUX video source for each pixel on the Amiga screen is that all even-numbered colors in the palette table correspond to the selection of one of AMUX's possible video sources and odd-numbered colors in the palette table correspond to the other AMUX choice.

Using a similar concept of how bitplanes #1 and #2 might combine to control the Fader level we can see that various combinations of 1-bits and 0-bits in those particular bitplanes will correspond to palette entries spaced at certain intervals in the palette. For example a 00 combination of bits in the #1 and #2 bitplanes correspond to palette entry positions evenly divisible by 4. The BitPlay animation merge utility actually does the positioning of palette positions in relation to bitplane bits slightly differently. Here's what BitPlay does:

Palette Positions	BitPlane 2 4-level Fader Control	BitPlane 1 4-level Fader Control	= Combined Fader Level	BitPlane 0 2-choice AMUX Selection
0 8 16 ...	0	0	= 0 (0%)	0
1 9 17 ...	0	0	= 0 (0%)	1
2 10 18 ...	0	1	= 1 (33%)	0
3 11 19 ...	0	1	= 1 (33%)	1
4 12 20 ...	1	0	= 2 (67%)	0
5 13 21 ...	1	0	= 2 (67%)	1
6 14 22 ...	1	1	= 3 (100%)	0
7 15 23 ...	1	1	= 3 (100%)	1

BitPlay merges the bitplanes used for AMUX and Fader control and generates a palette positioning order which is not strictly in increasing binary order - it pairs up possible Fader levels with both possible AMUX source levels. The pattern of 8 successive combinations repeats throughout the whole palette. Each Fader level appears twice in a row, with its two possible AMUX selector choices.

All the while the upper five bitplanes (which came from the original Amiga 32-color animation) keep the high order palette color values they had in the original Amiga animation. What's been done by BitPlay is to spread the original 32 colors related to the original 5 bitplanes into 256 colors by tacking on the 1-bitplane AMUX animation and the 2 bitplane Fader animation and the result is a palette in which there are patterns of 8 color variations which repeat essentially 32 times. Take a look at the color table entries in the CrUD source file for GIRAFFE (shown below) to see this pattern as BitPlay creates it for a real CrUD source file.

With that behind us (whew!) you could examine the frames of the GIRAFFE with a paint program and you'd find that certain areas of the images use pixels of colors which are appropriate. One of the disks accompanying this documentation (the GIRAFFE disk) actually contains the three original

animations used to construct the final GIRAFFE effect. You should examine all three of those animations. First look at the Giraffe.Latch animation to see how the single-bitplane animation reveals its role of broadly exposing either the Source2 video (to the left of the center of the Giraffe) or the Source1 video (to the right of the Giraffe). Then look at the Giraffe.Alpha animation to see the 2-bitplane animation follows the outline of the giraffe to let its edges smoothly fade between Source2 video and Encoder (the edges on the left side of the giraffe smoothly ramp over to the colored giraffe's Encoder signal) and the Encoder and Source2 on the right side edges of the giraffe. Then look at the Encoder animation to see the original 32-color giraffe animation itself. We'll run an example of creating this effect later and let BitPlay merge them together and then you can see the end result of the merge process and its effect on the original 32-color palette of the Giraffe.Encoder images.

Creating FX - FX Elements and The FX Environment Tools

FX Elements - CrUDs, Amiga Graphics, and Effects (Crouton) Icons

Each effect is built from (a mimimum of) three components, a Crouton User Data (CrUD) file and an Amiga Graphic file (either an ANIM or an ILBM IFF image), and an icon file for that effect. The CrUD defines that nature of the effect and how the Toaster will handle the companion Amiga IFF image or ANIM and run the effect. Amiga IFF graphics are already familiar. The icon file is simply a 4-color Amiga IFF ILBM of 80 x 50 pixels. A special tool (call RunNot) is used (automatically) during the effect-creation process to replace your 80x50 pixel IFF with an identical one which does not have the typical IFF run-length compression. The final icon used for your effect will be an uncompressed 80x50 IFF ILBM image.

A Closer Look at CrUD Source Files

Before moving on we should examine three CrUD source files for the examples of effects creation we'll be using. First let's look at the CrUD Source File for WIPE TWISTING NEON BANDS.

```
**************************************************************************
;CrUD source file for WIPE TWISTING NEON BANDS

;The following line is an assembler directive to merge in the tags.i assembly language include file of
;constant values and names in the course of the 'make' process conducted by SingleMake's call to the
;PhxAss assembler. C programmers need not worry - there's no assembly language to learn - the
;assembler is provided and called automatically by the SingleMake 'make' script.

    Include "tags.i"

    CrUD_START              CrUD_ILBMFX,CrUD_4_0

;The values of all the TAG_xxxx, AFXT_xxxx, and PACO_xxxx items are in the tags.i file. These
;particular tags and their values are long word 32-bit numbers, but not all NewTek effects tags are
;necessarily 32-bit.

;Each TAG_ declaration below is actually a macro with one parameter, see the tags.i include file.
;Some ;of the TAG_xxxx macros have numerical values and some are Boolean (TRUE or FALSE).

;TAG_FCountMode can be 0, 1, 2, or 3. The value here of 2 means variable speed is supported for
;this ;effect. The next few tags are supporting individual speed values used when there really is a
;variable ;speed.

    TAG_FCountMode          2
```

```
;TAG_VariableFCount is the variable frame count (speed) value.

    TAG_VariableFCount    40

;TAG_SlowFCount is the number of frames to be used when this effect is played at SLOW speed.

    TAG_SlowFCount        80

;TAG_MedFCount is the number of frames to play when this effect is in MEDIUM speed.

    TAG_MedFCount         40

;TAG_FastFCount is the number of frames to play for this effect in FAST speed.

    TAG_FastFCount        20

;TAG_Interlaced TRUE indicates the IFF ILBM used in this effect is interlaced.

    TAG_Interlaced        TRUE

;TAG_KeyMode here indicates that the Fader will be controlled by the DIB method (using 2 bits,
;four levels of Fader control between its AMUX and BMUX inputs). Note that the value
;AFXT_Key_4Level70ns is synonymous with AFXT_KeyDIB (see tags.i).

    TAG_KeyMode           AFXT_Key_4Level70ns

;TAG_Encoder TRUE means that Amiga graphics will appear on the BMUX for this effect. If we
;left this out or made it FALSE, then the BMUX would get Toaster Matte color generator input
;instead of Amiga graphics.

    TAG_Encoder           TRUE

;TAG_LatchAM TRUE means Amiga color palette bits will control the selection of video sources on
;the AMUX.

    TAG_LatchAM           TRUE

;TAG_ButtonELHlogic is set to a value which tells the Toaster software how to manipulate the
;behavior and appearance of the user interface buttons during the effect.

    TAG_ButtonELHlogic    AFXT_Logic_LatchAencoderB

;TABLE_PaletteColors is a structure macro which informs the effect how to manipulate and interpret
;Amiga palette colors during the effect.

    TABLE_PaletteColors

;The PACO_NumberOfPalettes tells the effect that a total of 340 different color combinations will
;occur during this effect. The effect starts with one 240-color palette but that palette is replaced 340
;times as successive colors in the table below are 'slid', one color at a time, into the Amiga palette for
;this image. The extra palette counts were inserted here as the effect was created, to compensate for
;the 'widths' of the several bands of color that create the soft edges. This allows the transitional
;values of the palette to cycle completely off the end of the color table, leaving the screen filled with a
;solid source.

    dc.w    240+12+32+12+32+12                    ;PACO_NumberOfPalettes
```

```
;This says the Amiga image palette colors will be replaced by shifting the whole palette by one
;position during the cycling process (a new top color enters from the next value in the table below).

    dc.w    1                   ;PACO_ColorsBetweenPalettes
                                ;number of colors between palettes
                                ;1 for moving pointer.

;The initial palette will start at the zero position of the table below.

    dc.w    0                   ;PACO_StartingColorNumber
                                ;Ususally 0

;240 colors at a time will be used in the Amiga image's palette.  Remember that for 256-color effects,
;the top 16 colors of the palette are reserved for internal use by the Toaster.

    dc.w    240                 ;PACO_NumberOfColors
                                ;Maximum of 240

;The following color table will be 'masked' into the Amiga image's palette in such a way that the bits
;in the color table's palette entries will be used as 'mask' values in the colors' low bits to let the
;Toaster use them as controls for the AMUX source (Latch) and four Fader (DIB) levels without
;changing the higher order bits that make up the palette values that are encoded into the colors used in
;the effect.

    dc.l    PACO_Mask_Latch!PACO_Mask_DIB

;This effect doesn't require any additional palette masking control values so the following three
;positions in this structure are set to 0 (PAC0_Mask_Nochange = 0, see tags.i.)

    dcb. l  3,PACO_Mask_Nochange                    ;other Masks not needed

;A block of 16 bytes at the end of a TABLE_PaletteColors structure is reserved for future use and
;filled with zeroes for now.

    dcb.b   16,0                                    ;space reserved for the future

;The following color table starts with a set of 240 bit-manipulation entries which ALL are set to start
;the effect by having the entire Amiga image's color palette 'latch' the user's current Main video bus
;selection onto the AMUX and have the Fader set to 100% AMUX level.  There are 240 long words
;here, each with the value of #$00000000 OR'd with the bit denoting Main bus 'latching' on AMUX.
;The #$00000000 value (0, zero) means that the Fader is at its zero position (100% on AMUX).
;Remember that this effect permits 4 levels of Fader control between AMUX video sources and the
;BMUX which is set to Encoder (Amiga graphics).  We start the effect with the Fader set to show
;only AMUX's source - and that source will be the user's selected Main video input.

* color table

    dcb. l  240,$00000000!PACO_CurrentMain

;As the color cycling begins, the first color which replaces the 'top' Amiga palette entry is
;immediately below - and it has AMUX still 'latched' to Main but now we've started to mix a small
;amount of the Amiga graphic by setting a value of 1 (#$00000001) to shift the Fader to 66.6%
;AMUX and 33.3% BMUX (which has Encoder selected).

    dcb.l   1,$00000001!PACO_CurrentMain
```

```
;The next two colors which get 'slid' down into the top of the Amiga image's palette also point the
;AMUX to the Main bus video input but now we've raised the Fader control level to 2 so we'll see a
;mix of 33.3% AMUX ('latched' to the Main ;video) and 66.6% Amiga graphics wherever this
;palette color appears.

    dcb.l    2,$00000002!PACO_CurrentMain

*-----------------------------

;Now we find a series of three colors which will slide down into the top of the Amiga image's palette
;in which the AMUX is still 'latched' to the Main video source, but we've now swung the Fader (the
;#$00000003 value) to 100% BMUX. That means that the Amiga image (Encoder source on BMUX)
;completely covers any part of the display where these three colors are displayed. Now our palette
;has elements where the bottom 234 colors still show 100% AMUX Main video but the top 6 colors
;show increasing levels of BMUX Encoder input. The top 6 colors 'ramp' up to show the Amiga
;graphic.

    dcb.l    3,$00000003!PACO_CurrentMain

;The following three colors to be slid down into the Amiga palette also point 100% to the BMUX, but
;we've switched the AMUX source (while it is really invisible for these colors) to the second video
;source (the replacement source to which the Toaster is switching).

    dcb.l    3,$00000003!PACO_CurrentOverLay

*-----------------------------

;You can now see that the rest of the colors which slide through the Amiga palette have varying
;combinations of effects. They successively manipulate combinations of AMUX video sourcing
;(Main and Overlay) and mixing of the AMUX source video with the Amiga graphic Encoder signal
;on BMUX.

;It is important to realize that the colors in this table don't really become full Amiga image palette
;entries and that the Amiga palette entries don't actually cycle at all. The values in this series of color
;table entries in the CrUD are just being used (sequentially) to readjust the effect of the low-order bits
;of each Amiga palette color on the Toaster's video mixing hardware.

    dcb.l    2,$00000002!PACO_CurrentOverLay
    dcb.l    1,$00000001!PACO_CurrentOverLay

    dcb.l    32,$00000001!PACO_CurrentMain

    dcb.l    1,$00000001!PACO_CurrentMain
    dcb.l    2,$00000002!PACO_CurrentMain

*-----------------------------
    dcb.l    3,$00000003!PACO_CurrentMain
    dcb.l    3,$00000003!PACO_CurrentOverLay
*-----------------------------

    dcb.l    2,$00000002!PACO_CurrentOverLay
    dcb.l    1,$00000001!PACO_CurrentOverLay

    dcb.l    32,$00000000!PACO_CurrentOverLay

    dcb.l    1,$00000001!PACO_CurrentMain
```

```
    dcb.l     2,$00000002!PACO_CurrentMain

*-----------------------------
    dcb.l     3,$00000003!PACO_CurrentMain
    dcb.l     3,$00000003!PACO_CurrentOverLay
*-----------------------------

    dcb.l     2,$00000002!PACO_CurrentOverLay
    dcb.l     1,$00000001!PACO_CurrentOverLay

;The effect ends by sliding in a complete set of control-bit replacements for all 240 colors in the
;Amiga palette - wherein the Source2 (CurrentOverlay) video shows everywhere and has completely
;replaced the original Main source

    dcb.l     240,$00000000!PACO_CurrentOverLay

;The following two macros are always found at the end of the CrUD source file, TABLE_END wraps
;up the TABLE_PaletteColors structure and CrUD_END wraps up the entire CrUD file.

    TABLE_END

    CrUD_END

*****************************************************************************
```

This is the CrUD source file for the CAMERA IRIS effect. This CrUD source file consists of a number of NewTek-defined Tag items and their values, along with a color table which contains color palette information for the Toaster to use at the time the effect is run. The Tag names and their allowable values are found in the tags.i assembler source file in the INC directory of the ENVIR drawer (see below). I've added a number of comments to this CrUD source file.

```
*****************************************************************************
```

;CrUD Source File for CAMERA IRIS

```
;The following line is an assembler directive to merge in the tags.i assembly language include file of
;constant values and names in the course of the 'make' process conducted by SingleMake's call to the
;PhxAss assembler.  C programmers need not worry - there's no assembly language to learn - the
;assembler is provided and called automatically by the SingleMake 'make' script.

    Include "tags.i"

;CrUD_START is a macro which has two constant value arguments.  The arguments specify what
;kind of effect and the Toaster software revision - in this case CrUD_ANIMFX means this CrUD will
;become part of an animation effect and CrUD_4_0 specifies Toaster 4.0 software usage.

    CrUD_START                CrUD_ANIMFX,CrUD_4_0

;The values of all the TAG_xxxx, AFXT_xxxx, and PACO_xxxx items are in the tags.i file.  These
;particular tags and their values are long word 32-bit numbers, but not all NewTek effects tags are
;necessarily 32-bit.  Each TAG_ declaration below is actually a macro with one parameter, see the
;tags.i include file.  Some of the TAG_xxxx macros have numerical values and some are Boolean
;(TRUE or FALSE).
```

```
;This indicates that this effect DOES have sound when the effect occurs at FAST speed (only) and its
;value (AFXT_AudioFast) indicates the sound is of unknown stereo or mono type.

    TAG_AudioFastSamples        AFXT_AudioFast

;This indicates that the effect will turn off the Amiga's audio filter

    TAG_TurnAudioFilterOff      TRUE

;This indicates this effect can be used on Amiga 2000 Toaster systems (non-AGA or non-AA
;systems).

    TAG_NonAAeffect             TRUE

;This item indicates that latching (AMUX or BMUX pixel-color-based video source selection) will be
;used in this effect for the BMUX.

    TAG_LatchBM                 TRUE

;This item specifies the color generated by the Toaster's Matte generator, in this case it will be pure
;black.

    TAG_MatteColor              AFXT_Matte_Black

;This item dictates the 'keying mode' - how the Fader will be operated under palette color control. In
;this case, the value of AFXT_Key_Fader means that the Fader will be under control of the CDAC 0-
;7 bits. For this particular effect, those bits don't change during the effect so this is simply one way
;of saying that no Fader operation takes place at all!

    TAG_KeyMode                 AFXT_Key_Fader

;The ButtonELHLogic tags specify how drawn buttons on the user-interface will behave during the
;effect. There are a number of combinations of how Toaster user interface buttons get enabled,
;disabled, and highlighted during and after an effect. This value is used to denote which combination
;of those button handling routines will be in force for this effect and also how the Toaster allocates
;various of its control bits to handle internal video signal routing and mux controls

    TAG_ButtonELHlogic          AFXT_Logic_TSEab

;The following macro sets up a specific format for a palette color data structure which will be
;interpreted by Toaster's effects-running system. Whereas tags are used as required to specify the
;nature of the effect, the following table always requires certain entries and will resemble this form in
;all CrUD source files. See tags.i for a complete description of the PaletteColors table. It also sets up
;the AMUX/BMUX/Fader control input routing in the Toaster.

    TABLE_PaletteColors

;The first entry in the table indicates the palette in use will be compatible with non-AA systems
;(A2000) and that one palette is used throughout the entire effect (we're not color cycling here!).

    dc.w    1                   ;PACO_NonAA_NumberOfPalettes

;The next entry sort of re-states that no cycling will take place, it means that there's a zero color
;cycling step size.

    dc.w    0                   ;PACO_NonAA_ColorsBetweenPalettes
```

```
;This denotes that the palette used by the effect will start with the first color in the color palette, the 0
;color.

    dc.w    0                       ;PACO_NonAA_StartingColorNumber

;This states how many colors are in the palette - only 4.

    dc.w    4                       ;PACO_NonAA_NumberOfColors

;This tells the Toaster how to process the values in the color table below. See the above discussion of
;how various bits in certain places in a color value can affect AMUX/BMUX source selection (known
;as 'latching'), etc. The color table provided below in the CrUD source file is deliberately
;construction without any real colors, but only certain bits activated which the Toaster will use to
;'mask' into the actual colors from the original animation. The masking method used by this effect
;means that all bit positions of the color values in the palette will be entirely REPLACED by the
;values specified in the following table. Since the tag entries for these values only contain the low-
;order control bits, the original palette from the anim will be replaced with a palette that appears
;entirely black on the Amiga RGB monitor but is filled with video sources or matte color on the Main
;Toaster output.

    dc.l    PACO_Mask_Replace

;There are three more long word slots in the PaletteColors table structure for additional MASKING
;information items. In this case, all three are simply set to 0 (the value of PACO_Mask_Nochange in
;the tags.i file) and are not with this effect.

    dcb.l   3,PACO_Mask_Nochange

;The PaletteColors table ends with a block of 16 bytes which are reserved for possible future ;use.

    dcb.b   16,0                    ;space reserved for the future

* color table

;The 0 palette color goes entirely unused in this animation effect.

    dc.l    $00000000

;The 1 palette color will be used to 'latch' the BMUX to whatever video source the user has selected
;on the Toaster's Main video bus when the effect occurs; the 'CurrentMain' source.

    dc.l    $00000000+PACO_CurrentMain

;The 2 palette color will be used to 'latch' the BMUX to whatever video source is set to replace the
;Main one, i.e. the source selected by the user on the Preview bus; the 'CurrentOverlay'.

    dc.l    $00000000+PACO_CurrentOverLay

;The 3 palette color will 'latch' the Toaster's Matte generator (already specified to be a black color by
;one of the tags earlier in this CrUD) to the BMUX. Note also that this value, PACO_EncoderMatte,
;means EITHER the Encoder OR the Matte - and which will be used is taken care of by a tag ;earlier
;in the CrUD. Matte is the default so no tag was actually used to specify this. Had we wanted the
;Encoder to be used for PACO_EncoderMatte, we'd have had to specify TAG_Encoder TRUE in the
;above set of tags for this CrUD.

    dc.l    $00000000+PACO_EncoderMatte
```

```
;Now two macros to end the CrUD's PaletteColors table structure, then finalize the CrUD (see tags.i)

    TABLE_END

    CrUD_END
*****************************************************************************
```

This is the CrUD source file for the GIRAFFE effect, our third example. A more limited set of comments is applied to this example since a number of the TAG_xxxx items and their values were already explained above.

```
*****************************************************************************
;CrUD source file for GIRAFFE

    Include "TAGS.I"

    CrUD_START              CrUD_ILBMFX,CrUD_4_0

;Encoder will show on the BMUX

    TAG_Encoder             TRUE

;This does not play as a variable speed effect.

    TAG_FCountMode          1
    TAG_VariableFCount      0

;SlowFCount is 0 so no SLOW speed; MedFCount is 2 so each giraffe animation frame plays 2
;frames in MEDIUM speed; FastFCount is 1 so each giraffe frame plays 1 frame in FAST speed.

    TAG_SlowFCount          0
    TAG_MedFCount           2
    TAG_FastFCount          1

    TAG_ButtonELHlogic      AFXT_Logic_LatchAencoderB

;'Latching' (video source selection) on AMUX, keying (Fader control) controlled by DIB (same as
;4Level70ns, see tags.i).

    TAG_LatchAM             TRUE
    TAG_KeyMode             AFXT_Key_DIB

;If you want sound present on FAST speed play (its sample file should be in Speed1 drawer of csrc
;during effect creation by SingleMake, see below). These tags indicate that sound is present on FAST
;speed but I'm informed there really isn't a sound for GIRAFFE and that these tags simply are
;ignored because no sound file is available in csrc/Sound1 of the Envir drawer (see below).

    TAG_AudioFastSamples    AFXT_AudioFast
    TAG_TurnAudioFilterOff  TRUE

    TABLE_PaletteColors

    dc.w    1
    dc.w    0
    dc.w    0
```

```
    dc.w    240
    dc.l    PACO_Mask_DIB+PACO_Mask_Latch
    dcb.l   3,PACO_Mask_Nochange
    dcb.b   16,0

;Every other color alternates the AMUX 'latch' source selection and successive pairs of colors will
;have varying levels of Fader mixing between AMUX and Encoder present on BMUX.  PACO_DIB0
;= 0, PACO_DIB1 = 1, PACO_DIB2 = 2, and PACO_DIB3 = 3 (the four levels of DIB-controlled
;Fader level).  In one of the earlier CrUD examples we saw these DIB levels explicitly declared as
;numerical values (#$00000000, #$00000001, #$00000002, and #$00000003) while here they are
;declared using value names from tags.i.

* ---------- Color Table with 240 entries --------

    dc.l 0+PACO_DIB0+PACO_CurrentOverLay
    dc.l 0+PACO_DIB0+PACO_CurrentMain
    dc.l 0+PACO_DIB1+PACO_CurrentOverLay
    dc.l 0+PACO_DIB1+PACO_CurrentMain
    dc.l 0+PACO_DIB2+PACO_CurrentOverLay
    dc.l 0+PACO_DIB2+PACO_CurrentMain
    dc.l 0+PACO_DIB3+PACO_CurrentOverLay
    dc.l 0+PACO_DIB3+PACO_CurrentMain

    .
    .
    .

    (the pattern of 8 values above repeats 28 times here)

    .
    .
    .

    dc.l 0+PACO_DIB0+PACO_CurrentOverLay
    dc.l 0+PACO_DIB0+PACO_CurrentMain
    dc.l 0+PACO_DIB1+PACO_CurrentOverLay
    dc.l 0+PACO_DIB1+PACO_CurrentMain
    dc.l 0+PACO_DIB2+PACO_CurrentOverLay
    dc.l 0+PACO_DIB2+PACO_CurrentMain
    dc.l 0+PACO_DIB3+PACO_CurrentOverLay
    dc.l 0+PACO_DIB3+PACO_CurrentMain

    TABLE_END

    CrUD_END

*****************************************************************************
```

The FX Creation Environment and Tools

An automated processing system for building effects from the three component parts is part of your developer package. It is called FXTools and comprises a directory on an Amiga storage volume. FXTools includes a number of programs (tools) and subdirectories.

The tools in FXTools are:

BitPlay - a program which merges 3 anim files into one and creates an appropriate CrUD file for inclusion in the final effect. BitPlay is called automatically by the animated effects 'make' script. The CrUD file created by BitPlay is actually an assembly language source file for the CrUD chunk of an effect file.

AnimTool - a program which can process an existing anim file and convert it to a specialized form (a Smart Anim) which plays at maximum speed under the Toaster's FX anim playing system. Smart Anim files may be mixtures of different types of Amiga anim compression. AnimTool analyzes the anim file it is processing and determines what form of frame-to-frame compression is best for each successive frame. The result is a Smart Anim which partly use Anim5, Anim7, or Anim8 compression between frames - or even no compression at all between certain frames. AnimTool also analyzes your original animation to detect whether it can play smoothly on a Toaster effect. For most of your original animations it is wise to generate them as 384x241 with 32 colors. Many animations of this type can simply be prepared as Anim7 or Anim8 files and won't require conversion to Smart Anim format. You can test the resulting Toaster animated effects yourself and determine whether they'd benefit from AnimTool processing. AnimTool is not called automatically by the batch AmigaDos script files that build the effect from its components. AnimTool is supplied for you to use as needed.

AnimSplicer - a animation construction and deconstruction utility. AnimSplicer, like AnimTool, is not called automatically by the effects 'make' scripts. It is provided to let you merge animations you've built in sections into a single Anim file.

The subdirectory structure of FXTools looks like this:

Make4.0 - drawer
- csrc - drawer containing source portions of your effect
  - Anim - source ILBM or ANIM files (SingleMake looks here)
    - Date - drawer used for temporary date storage and compares
  - CrUD - drawer containing CrUD files corresponding to the contents of the Anim drawer's graphics files (SingleMake looks here for the CrUD source file)
    - Date - drawer used for temporary date storage and compares
    - Binary - drawer for CrUD object file binary data
  - Icon - drawer containing your crouton icon IFF images
    - Has icons for the example effects creations we'll do plus a sample of a color icon (VTLogo), and the 'GenericIcon'.
    - Date - drawer used for temporary date storage and compares
  - Sound1 - drawer with IFF 8SVX sampled sound for FAST speed
    - Camera Iris sound effect file
    - Date - drawer used for temporary date storage and compares
  - Sound2 - drawer with IFF 8SVX sampled sound for MEDIUM speed
    - Date - drawer used for temporary date storage and compares
  - Sound3 - drawer with IFF 8SVX sampled sound for SLOW speed
    - Date - drawer used for temporary date storage and compares

Toaster - drawer
Effects - drawer where final effect and icon are deposited by the SingleMake 'make' script
Envir - drawer
Inc - has NewTek assembly include files required for the assembler to process the CrUD files during the 'make' process
Exec - drawer with a very few Amiga Exec assembler include files

Bat - has batch files related to effects construction including SingleMake (the one we'll use). SingleMake will perform all the necessary steps to build a usable effect from its components.

Bin - binary files (various programs and tools) used by the SingleMake 'make' script, including:

AddPus - appends the Parse Upward Search structure to the CrUD
AnimLen - determines # of frames in an Anim file
Append - adds CrUD data file to the effect and icon files
CmpDates - checks to see if dates on effect and icon match
iff.library - gets copied to your libs: directory (if needed)
MatchDate - makes dates of effect and icon identical
Obj2Data - converts assembled CrUD object files to CrUD data
PhxAss - a P.D. assembler which assemble CrUD source files
RunNot - a program which converts an 80x50 pixel typically compressed IFF ILBM image into a form (non-compressed ILBM) used by the Toaster as a crouton icon. RunNot is called automatically by the SingleMake 'make' script.
RunNot_CAMG - a close relative of RunNot which adds a CAMG, obsolete

These are utilities in bin which may also be useful but are not called by SingleMake:

StompCMAP - utility to replace a ILBM's CMAP with the StompList
StompList - shows format for list of transparency values
SwapCMAP - CMAP in an ILBM with one from another ILBM

A Trial Run with SingleMake

The Envir and Make4.0 drawers are already organized for use with SingleMake. Let's run an 'thought' example 'make' to create an effect from its components. All you have to do to create an effect is first install the FXTools 3-disk set as provided. The Install

First puts the setup file in your s: directory
Assigns nd2: to work:FXTools
Puts the rest of the directories and files in appropriate place on your Work: partition.

Then:

0. Decide on a name for your effect, let's use SAMPLE
1. Put your Anim or ILBM file into Make4.0/csrc/Anim/SAMPLE
2. Put the corresponding CrUD file in Make4.0/csrc/CrUD/SAMPLE
3. Put the sound file for the effect's FAST speed (if any) into Make4.0/csrc/Sound1/SAMPLE
4. Put the sound file for the effect's MEDIUM speed (if any) into Make4.0/csrc/Sound2/SAMPLE
5. Put the sound file for the effect's SLOW speed (if any) into the Make4.0/csrc/Sound3/SAMPLE
6. Put the icon ILBM you've designed for this effect into Make4.0/csrc/Icon/SAMPLE. If you don't

have your own icon designed, the SingleMake script will use a ready-made default icon already stored in Make4.0/csrc/Icon.

7. Double-click on the Amiga Default Icon for SingleMake, and Workbench will present the 'Execute Command' window with SingleMake already in its string gadget, so just add the name of the effect to the string gadget (e.g. SingleMake SAMPLE) and hit 'return'. You'll wind up with an effect and its icon in the

   Work:FXTools/Make4.0/Toaster/Effects directory with the names: SAMPLE and SAMPLE.I

Here's the sequence of events which takes place when you execute the SingleMake script. In fact, here's the SingleMake script itself, all commented.

```
**************************************************************************
;Commented version of the SingleMake script!

;read in the file name selected by the user on the Shell command line

.k file

;call the setup script to temporarily assign names for various source and destination directions of bits
;and pieces of the effect.

s:fxsetup

;**************************************************************************
;* Make a single croutine.     By SKell,  ©NewTek Inc.  Feb 1994
;**************************************************************************

;tell user what's up and what effect name is being processed now

echo "=========================================================================="
echo "   Processing $VTCR/<file>"
echo "--------------------------------------------------------------------------"
echo >>$TEMPDIR/CROUTON.err "   Processing $VTCR/<file>"

;following line only required if using the HiSoft assembler (which isn't included with this developer
;pack - it's a commercial assembler you can purchase)

;echo >$TINC/HiSoftPath.i " INCDIR '$AINC/'"

;-------------------------

FAILAT 10

;are dates of the current CrUD source and the CrUD binary file different? i.e. - is the CrUD source
;really new?

    $MBIN/cmpdates >nil: "$CSRC/<file>" "$CSRC/binary/<file>"

;if so, then we need to build a new effect and icon

    IF WARN
```

```
;first delete the old ones
        execute $MBAT/IfDelete "$VTCR/<file>"
        execute $MBAT/IfDelete "$VTCR/<file>.I"
        wait 1
        copy "$CSRC/<file>" $TEMPDIR/asm1.temp

;assemble the CrUD source file in ram
        FAILAT 200
            $MBIN/PhxAss $TEMPDIR/asm1.temp -I$TINC/,$AINC/ -O$TEMPDIR/asm2.temp

        IF WARN
            SKIP ErrorHandler
        ENDIF
    FAILAT 10

;convert the resulting CrUD object file into binary data
    $MBIN/obj2data $TEMPDIR/asm2.temp "$CSRC/binary/<file>"
    delete $TEMPDIR/asm?.temp QUIET

;and shove the date from the CrUD source onto the CrUD binary data file
    echo "Match 1"
    $MBIN/matchdate "$CSRC/<file>" "$CSRC/binary/<file>"
ENDIF

;-------------------------

;see if there is an anim file for this effect in csrc/Anim

IF EXISTS "$ASRC/<file>"
    FAILAT 10

;if so, see if the dates match between the anim file you want to use and the date last given when this
;anim file was used in building an effect

    $MBIN/cmpdates >nil: "$ASRC/<file>" "$ASRC/date/<file>"
    IF WARN

;if the csrc/anim file is newer, get rid of the old effect and

        execute $MBAT/IfDelete "$VTCR/<file>"

;run NumFields on this anim to measure its length

        execute $MBAT/NumFields "$ASRC/<file>"
        wait 1

;now slam the date from the newer csrc/anim as the anim current date

        echo "Match 2"
        $MBIN/matchdate "$ASRC/<file>" "$ASRC/date/<file>"
    ENDIF
ENDIF

;-------------------------
```

```
;is there an icon file for this effect in csrc/icon?

IF EXISTS "$ISRC/<file>"
    FAILAT 10

;if so, see if the dates match between the icon file you want to use and the date last given when this
;icon file was used in building an effect

    $MBIN/cmpdates >nil: "$ISRC/<file>" "$ISRC/date/<file>"

;if the csrc/icon file is newer, get rid of the old effect and icon

    IF WARN
        execute $MBAT/IfDelete "$VTCR/<file>"
        execute $MBAT/IfDelete "$VTCR/<file>.I"
        wait 1
        copy "$ISRC/<file>" $TEMPDIR/runnot.temp QUIET

;and run the RunNOT program on it to decompress its ILBM

        $MBIN/RunNOT "$ISRC/<file>" $TEMPDIR/runnot.temp
        copy $TEMPDIR/runnot.temp "$ISRC/<file>" QUIET
        delete $TEMPDIR/runnot.temp QUIET

;now slam the date from the newer csrc/anim as the icon current date

        echo "Match 3"
        $MBIN/matchdate "$ISRC/<file>" "$ISRC/date/<file>"
    ENDIF

;if there was no specific icon ready to go, use the generic icon in csrc/icon (ready to go for you for
;rapid prototyping without having to set up an specifically named icon file in csrc/icon each time

ELSE
    FAILAT 10

;you get the picture - check date match between what exists and what you want - delete unproductive
;and unrelated earlier stuff if necessary before making new.  Do RunNOT if necessary on the generic
;one, etc. and reset the current date for icon making.

    $MBIN/cmpdates  >nil: "$ISRC/Generic.icon" "$ISRC/date/Generic.icon"
    IF WARN
        execute $MBAT/IfDelete "$VTCR/Generic.icon"
        execute $MBAT/IfDelete "$VTCR/Generic.icon.I"
        wait 1
        copy "$ISRC/Generic.icon" $TEMPDIR/runnot.temp QUIET
        $MBIN/RunNOT "$ISRC/Generic.icon" $TEMPDIR/runnot.temp
        copy $TEMPDIR/runnot.temp "$ISRC/Generic.icon" QUIET
        delete $TEMPDIR/runnot.temp QUIET
        echo "Match 4"
        $MBIN/matchdate  "$ISRC/Generic.icon" "$ISRC/date/Generic.icon"
    ENDIF

ENDIF

;-------------------------
```

```
;now do a similar sequence of date checking, delete old if necessary, use new if appropriate, etc. and
;match date when done - all on the sound files for this effect.  Sound1, Sound2, and Sound3 sound
;sources (FAST, MEDIUM, and SLOW speed sound files) are all handled below similarly.  Note that
;this script doesn't fail or moan if the sound files are entirely absent - again for convenience when
;prototyping - but does handle them properly if they're there for a final 'make'

IF EXISTS "$SSRC1/<file>"
    FAILAT 10

    $MBIN/cmpdates >nil: "$SSRC1/<file>" "$SSRC1/date/<file>"
    IF WARN
        execute $MBAT/IfDelete "$VTCR/<file>"
        wait 1
        echo "Match 5"
        $MBIN/matchdate  "$SSRC1/<file>" "$SSRC1/date/<file>"
    ENDIF
ENDIF

;-------------------------
IF EXISTS "$SSRC2/<file>"
    FAILAT 10
    $MBIN/cmpdates >nil: "$SSRC2/<file>" "$SSRC2/date/<file>"
    IF WARN
        execute $MBAT/IfDelete "$VTCR/<file>"
        wait 1
        echo "Match 6"
        $MBIN/matchdate  "$SSRC2/<file>" "$SSRC2/date/<file>"
    ENDIF
ENDIF

;-------------------------
IF EXISTS "$SSRC3/<file>"
    FAILAT 10
    $MBIN/cmpdates >nil: "$SSRC3/<file>" "$SSRC3/date/<file>"
    IF WARN
        execute $MBAT/IfDelete "$VTCR/<file>"
        wait 1
        echo "Match 7"
        $MBIN/matchdate "$SSRC3/<file>" "$SSRC3/date/<file>"
    ENDIF
ENDIF

echo "-----------------------------------------------------------------"

; now we take all the pieces (ANIM or ILBM, ICON, SOUND1, SOUND2, and SOUND3, and
;finally the CrUD data) and APPEND them all together to create an final EFFECT!

IF NOT EXISTS "$VTCR/<file>.I"

 IF EXISTS "$ISRC/<file>"
  copy "$ISRC/<file>" "$VTCR/<file>.I"
 ELSE
  copy $ISRC/Generic.icon "$VTCR/<file>.I"
 ENDIF
  $MBIN/append >nil: "$CSRC/binary/<file>" "$VTCR/<file>.I"
ENDIF
```

```
IF NOT EXISTS "$VTCR/<file>"

 IF EXISTS "$ASRC/<file>"
  copy "$ASRC/<file>" "$VTCR/<file>"
 ENDIF

 IF EXISTS "$SSRC1/<file>"
  $MBIN/append >nil: "$SSRC1/<file>" "$VTCR/<file>"
 ENDIF

 IF EXISTS "$SSRC2/<file>"
  $MBIN/append >nil: "$SSRC2/<file>" "$VTCR/<file>"
 ENDIF

 IF EXISTS "$SSRC3/<file>"
  $MBIN/append >nil: "$SSRC3/<file>" "$VTCR/<file>"
 ENDIF

 $MBIN/append >nil: "$VTCR/<file>.I" "$VTCR/<file>"

;and tell the user all is finished and well

echo "$VTCR/<file> UPDATED !!!!!!!!!!!!!!"
echo >>$TEMPDIR/CROUTON.err "$VTCR/<file> UPDATED !!!!!!!!!!!!!!"

;-----------------------------------

;if the original dates matched and there really was no work for this 'make' script to do, then say so

ELSE
 echo "$VTCR/<file> UNCHANGED.  It was up-to-date."
 echo >>$TEMPDIR/CROUTON.err "$VTCR/<file> UNCHANGED.  It was up-to-date."
ENDIF

echo "=============================================================================="
echo ""
echo >>$TEMPDIR/CROUTON.err ""
skip AllDone

************************************

;go here if the assembler fails to process the CrUD file

LAB ErrorHandler
      execute $MBAT/ErrorBeep
      quit 20
LAB AllDone

;that's all folks - end of the SingleMake script

*******************************************************************************
```

Summary of FX Creation and SingleMake

All the pieces of several effects are provided in the Make4.0 directory and its subdirectories. You can see that when you create an effect, you give all the parts (the CrUD, the ILBM or ANIM, the SOUND1, SOUND2, and SOUND3, and the ICON parts) all identical names. The SingleMake script looks to see if there are matching 'last created date' entries for your effect in the following subdirectories of the effects environment drawer (using the CmpDates program in FXTools/Envir/bin):

Make4.0/Toaster/Effects/name	FXcrouton date
Make4.0/Toaster/Effects/name.i	crouton icon date
Make4.0/csrc/CrUD/Date/name	crouton data file date
Make4.0/csrc/ANIM/Date/name	anim/ilbm file date
Make4.0/csrc/SOUND1/Date/name	sound1 file date
Make4.0/csrc/SOUND2/Date/name	sound2 file date
Make4.0/csrc/SOUND3/Date/name	sound3 file date
Make4.0/csrc/ICON/Date/name	icon image file date

If matching dates aren't found, SingleMake assumes it needs to create the currently named effect from the parts found in:

Make4.0/csrc/CrUD/name	CrUD source file
Make4.0/csrc/ANIM/name	anim/ilbm file
Make4.0/csrc/SOUND1/name	sound1 file
Make4.0/csrc/SOUND2/name	sound2 file
Make4.0/csrc/SOUND3/name	sound3 file
Make4.0/csrc/ICON/name	icon IFF ILBM file

and it updates the Date subdirectories as it goes, then puts the final results in FXTools/Make4.0/Toaster/Effects. SingleMake also updates the Date subdirectories with exactly matching date stamp files (using the MatchDate program in the FXTools/Envir/bin directory) so if you try to re-do exactly the same effect creation nothing will happen (because all the dates match). SingleMake will only re-do those parts of its job which it thinks it needs to by finding mis-matched dates on components of the effect.

FX Creation Examples

There are three example FX creation in the Make4.0 drawer which areready for 're-creation' with SingleMake.

1. Wipe Twisting Neon Bands

This effect has a Make4.0/csrc/CrUD drawer source file (we got a detailed look at that earlier), an ILBM single image in the Make4.0/csrc/ANIM drawer, and an ICON image IFF ILBM in the Make4.0/csrc/ICON drawer. To re-create this effect, simply open the Envir/Bat drawer, tell the Amiga Workbench to 'Show All Files', then double-click on the SingleMake icon. The Workbench will present its Execute Command window with SingleMake already in the string gadget. Just make the string gadget say:

SingleMake "Wipe Twisting Neon Bands"

and hit 'return'. Remember, you need a space character after SingleMake and you do need the quote characters because this file name has spaces in it. You can observe the process by which SingleMake dces its file date comparing, assembly of the CrUD file by the PhxAss assembler, conversion of the CrUD assembled object file to binary data only with the Obj2Data program, processing of the 'anim'

(in this case only a single ILBM image - and SingleMake will say so!) with the AnimLen program, processing of the icon ILBM IFF image into an uncompressed IFF file with the RunNot program, forced matching of dates with the MatchDate program, etc.

All of the NewTek programs used by the SingleMake script (e.g., CmpDates, MatchDates, Obj2Data, RunNot, PhxAss, AnimLen, etc.) are in the FXTools/Envir/bin drawer.

If you execute the SingleMake "Wipe Twisting Neon Bands" as instructed above, you'll observe the SingleMake script in action and when it's finished you can look in the FXTools/Toaster/Effects drawer and find the finished "Wipe Twisting Neon Bands" effect crouton file and its icon.

2. and 3. Camera Iris and Giraffe

You can try the same procedure on the Camera Iris and Giraffe effects examples, which are also pre-arranged with their respective components ready-to-go in the respective Make4.0/csrc/xxx drawers. Note that Camera Iris has a true ANIM file in the Make4.0/csrc/ANIM drawer (unlike Neon Bands, which was just a single image), and it also has a sound file in the Sound1 drawer. Of course, Camera Iris, like alleffects-under-construction, has a CrUD source file in the Make4.0/csrc/CrUD drawer, too (also detailed earlier).

Giraffe has a full animation, too. I've also put the original parts of the Giraffe animation (the Latch, Encoder, and Alpha separate animations) in the Make4.0/csrc/ANIM drawer for you to examine and experiment with using BitPlay and AnimTool (in the root of the FXTools drawer) later. But for the time being you can simple go through a trial 're-creation' of the Giraffe effect, too, using the technique shown above to activate

SingleMake "Camera Iris"

and

SingleMake Giraffe

with the Workbench's Execute Command window and string gadget.

Rolling Your Own - Making Your Own 'Neon Bands' Effect

Now that you see what pieces are required to construct an effect crouton and its icon, you can try substituting your own color cycling image for the one in the Twisting Neon Bands effect. Here's what you need:

1. Make a 768x480 256-color AGA image using only the first 240 colors in the 256-color palette (leave colors 240-255 as black).
2. Save your image with the name MyImage into the Make4.0/csrc/ANIM drawer.
3. Make a miniature, 80x50 pixel 4-color version of your image, perhaps with DeluxePaint or some other program which lets you re-size your image or cut it as a brush and then re-size it..
4. Save the miniature, 80x50 pixel 4-color version with the name MyImage into the Make4.0/csrc/ICON drawer.
5. Make a copy of the CrUD source file we already have available, namely, the "Wipe Twisting Neon Bands" CrUD (from the Make4.0/csrc/CrUD drawer) and name it Make4.0/csrc/CrUD/MyImage (just re-create the CrUD with a new name).
6. Now you have the essential components required to make your own effect, named MyImage, which will work quite like the 'Wipe Twisting Neon Bands' effect.
7. Run the SingleMake MyImage command as shown earlier for the other examples (using the Workbench Execute Command string gadget to set it up)

8. When SingleMake is finished, just copy the finished effect crouton file and its corresponding icon file (Make4.0/Toaster/Effects/MyImage and Make4.0/Toaster/Effects/MyImage.i files) to your 'real' Toaster Effects drawer and try it out!

When you run your MyImage effect you'll probably notice it doesn't work quite as smoothly and nicely as the real 'Wipe Twisting Neon Bands' effect does. That's because the original was developed so its color palette and CrUD Color Table were closely matched - making the sliding color masking toaster source control bits perfectly designed in the CrUD for that original color palette. You can try to perfect your MyImage effect further by trying a couple of things:

1. Try 'stealing' the color palette from the original 'Wipe Twisting Neon Bands' image file in the Make4.0/csrc/ANIM drawer and apply that exact color palette to your MyImage picture and re-do the SingleMake process.

2. Try studying the detailed impact of specific color palette masking control tag items in the original 'Wipe Twisting Neon Bands' CrUD file and re-arrange them as you think they may better apply to the palette you used in your MyImage picture, then re-create the effect crouton again with SingleMake.

Making Your Own Camera Iris-like Effects

You should be starting to get the picture of how to use the FXTools environment to make FX croutons which are analogous to the three examples provided here. If you want to make something like the Camera Iris, simply create your own ANIM which is a 4-color anim file sort of like the Camera Iris (it can even be one more like a KiKi effect as those are quite similar and use a similar CrUD source file). Save the appropriate pieces (an ANIM, a CrUD, and an ICON) in the appropriate drawers and run the SingleMake on YOUR effect. Remember to keep all the names identical for the component parts. You could make an animation of four blocks which move in from the corners of the screen and then back away into the same corners again, or you could make an animation which moves across the screen like a KiKi effect 'shadow'. The possibilities here are pretty diverse.

Making Your Own Giraffe-like Effects

These are a bit more complicated, but they follow the same principles as the other 'roll-your-own' examples above. Before you try this one yourself, you really need to examine the 3 component animations of the original Giraffe effect to see how they work and how you might 'imitate' them for your own work. Brilliance (from Digital Creations) has excellent animation loading and playing tools to let you examine the component animations in detail. You'll find the original Giraffe component animations in Make4.0/csrc/ANIM/Giraffe.Latch, Giraffe.Alpha, and Giraffe.Encoder. To see how BitPlay 'merges' them together into a single final anim file for use in creating the actual effect, see the Make4.0/csrc/ANIM/Giraffe animation file by loading it into Brilliance, too. To create a proper animation for a high-color Giraffe-like effect:

1. Make a 32-color animation and DO NOT USE THE TOP TWO COLORS (leave colors #30 and #31 black, and don't use them in your animation imagery). This is your '.Encoder' animation file.

2. Make a 2-color animation (black and white) which follows the main character of your '.Encoder' animation across the screen. If you intend to use the same CrUD file 'ripped' off from the Giraffe effect, you'll find that where you use the 0-color in the 'latch' animation is where the 'main' source video will show and where you use the 1-color in this 'latch' animation is where the 'new' source video will show. You should arrange that the 'break' from 0-color to 1-color lies in the center of your 'Giraffe', and not simply along one edge.

3. This part is harder. You make a 4-color animation which lets the video sources smoothly ramp up from the 'new/overlay' source video into your 'Giraffe' along its left edge and then smoothly ramp down from the 'Giraffe' right edge into the 'old/main' source video. The purpose of this '.Alpha' animation is to smoothly vary the transparency and smooth the edges around your 'Giraffe' (or whatever it is that plays the same role as the good old Giraffe did). Here's where you'll really benefit from closely examining the Giraffe.Alpha animation file (in Make4.0/csrc/ANIM) with a program like Brilliance so you can see which of the four colors plays each of the transparency level roles in the real Giraffe effect.

If you use Brilliance to examine all three of these component anims and then to view the end result anim you'll get the full picture of the role each component plays in constructing the full anim used in the effect (done with BitPlay). Like we did with the other two examples, you can copy the CrUD source file outright from the Giraffe effect and use it for your own 'analog' of the Giraffe. You'll have to be a bit more careful in constructing the anim than you had to be with Camera Iris (a much simpler type of effect) but there's no reason you can't create highly colorful animated transition effects along the lines of Giraffe after you master the art of making component animations and merging them with BitPlay (and possibly fine-tuning them with AnimTool).

Using BitPlay

The BitPlay program is in the root directory of your FXTools drawer. BitPlay merges the component animations of a Giraffe-like effect (the 'latch', 'alpha', and 'encoder' animations) into a final animation for creating the effect. BitPlay can actually run the whole process, including the creation of a CrUD source file, creation of the final effect animation, and execution of the SingleMake script to complete the whole job.

BitPlay is pre-set to expect a few 'assigned' Amiga volume names:

W:	the root directory in which the animation component subdirectories are found
W:L	the subdirectory for the 'Latch' animation
W:A	the subdirectory for the 'Alpha' animation
W:E	the subdirectory for the 'Encoder' animation
Process:	the directory in which BitPlay can find the AnimSplicer program
ASRC:	the directory into which you wish the completed effect animation to be stored
MBAT:	the directory in which BitPlay can find the MakeEffect script (just a copy of the SingleMake script described earlier, BitPlay is 'hard-coded' to look for MakeEffect)
DTBL:	the directory in which the CrUD source file is to be created by BitPlay

I've provided a little script, DoBitPlay, which performs these assigns appropriately for our test and experimentation purposes as we re-create the Giraffe effect starting from 'scratch' (starting, that is, from only the component animations and a pre-created effect Crouton icon from the existing Giraffe effect file). Here is the DoBitPlay script (as provided on the updated NTDevCon disk's 's' directory):

```
**********************************************************************
DoBitPlay Script - by Daniel Wolf 12/7/94

Assign w: ram:
Assign Process: Work:FXTools
Assign ASRC: work:fxtools/make4.0/csrc/Anim
Assign DTbl: work:fxtools/make4.0/csrc/CrUD
Assign MBAT: work:fxtools/envir/bat

Failat 20

makedir w:d
makedir w:a
makedir w:e
makedir w:l

copy work:fxtools/make4.0/csrc/anim/Giraffe.Encoder w:e
copy work:fxtools/make4.0/csrc/anim/Giraffe.Latch w:l
copy work:fxtools/make4.0/csrc/anim/Giraffe.Alpha w:a

work:fxtools/bitplay
**********************************************************************
```

Note that I've made life simple with this script by letting most of the BitPlay stuff take place in the Amiga ram disk. You'll probably need a couple of megabytes or so of ram.

Let's go through a test run of BitPlay and let it build a B-version of the Giraffe Effect from the component animations.

1. Open a Shell window.
2. Type in: DoBitPlay (and hit 'Return')
3. You'll now see the BitPlay program's window open and present you with a series of string gadgets which need to be filled.
4. Click on the question mark button next to the string gadget labelled Alpha
5. The file requester will appear (pre-set to the W:A directory) and you should select the file 'Giraffe.Alpha' and then click on the OK button of the file requester.
6. Click on the question mark button next to the string gadget labelled Encoder
7. The file requester will appear (pre-set to the W:A directory) and you should select the file 'Giraffe.Encoder' and then click on the OK button of the file requester.
8. Click on the question mark button next to the string gadget labelled Latch
9. The file requester will appear (pre-set to the W:A directory) and you should select the file 'Giraffe.Latch' and then click on the OK button of the file requester.
10. Click on the button labelled AA (this will be an AA effect for AGA Amigas only)
11. Click in the string gadget at the top labelled Current Effect
12. Type in: GiraffeB (and hit return)
13. You'll see immediately that the string gadget labelled Output (at the bottom) gets filled in automatically and now says: W:GiraffeB
14. There's one more string gadget of interest, labelled Data Table. If you leave this blank, BitPlay will create a completely fresh CrUD file for GiraffeB in the DTbl: directory (which we've conveniently assigned to be the work:FXTools/make4.0/csrc/CrUD directory - precisely where we want it). If instead you click on the question mark button next to this string gadget you can select a pre-existing CrUD file from the DTbl: directory and BitPlay won't create its own new one. Leave this string gadget blank so BitPlay will create a new CrUD source file named GiraffeB.
15. Now simply click on the GO button at the bottom of the BitPlay window.
16. BitPlay now goes into action. It reads in the three component animations and creates, frame by frame, the individual 8-bitplane composited frames needed to build a Giraffe effect animation with

the appropriately sorted bitplanes (see lengthy discussion above on bitplane sorting for colored animation effects). This may take a few minutes as BitPlay must process 120 frames for the Giraffe effect.

17. When BitPlay itself finishes the creation of the individual composited 8-bitplane frames, BitPlay calls on the AnimSplicer program from the Process: directory (also located in the FXTools directory root and conveniently pre-assigned as Process: in the DoBitPlay script) to stitch all the frames back together into a final Giraffe animation. AnimSplicer will then put the result, the GiraffeB effect anim file, into the ASRC: direcotory (which we've already conveniently assigned to be the work:FXTools/make4.0/csrc/Anim directory - just where we need it).
18. Now BitPlay calls on the MakeEffect script (just a copy of the SingleMake script in the MBAT: drawer, conveniently pre-assigned by DoBitPlay as the work:FXTools/envir/bat drawer) and goes through the same set of steps we've already discussed (above) to check dates, assemble the CrUD object file, and merge the newly created anim file with the CrUD, etc to create a finished GiraffeB effect. The end result, just as with our earlier experiments with SingleMake, is a finished GiraffeB effect and its corresponding icon file, GiraffeB.I, in the work:FXTools/make4.0/Toaster/Effects drawer.

Note that if you really want a nice icon for the GiraffeB effect, you should copy the original Giraffe effect's icon file (copy work:FXTools/make4.0/csrc/Icon/Giraffe.I to work:FXTools/make4.0/csrc/icon/GiraffeB.I) before you do the BitPlay GO. Don't worry, if you don't do that, remember that the MakeEffect script (same as SingleMake) will use a default icon (the Generic icon) which is already there in the Icon directory.

With the completed effect and its icon now in hand, you can simply copy them to the Effects/Animals drawer of your Toaster system and try them out!

NOTE: The version of BitPlay supplied appears to make slightly 'buggy' CrUD files. An update should be available soon, but in the meantime you may wish to re-create the effect with SingleMake (using a copy of the real Giraffe CrUD file supplied here) AFTER BitPlay has done it. The final effect anim created by this version of BitPlay does appear to be correct, so for the time being, BitPlay's main use is to merge the component animations only.

AlgoFX For the Video Toaster

by Daniel Wolf

10/26/95

Digital video switching effects (DVEs) for the Video Toaster encompass a very large subset of the available and interesting Toaster switching effects. What sets them apart from the other classes of Toaster effects is the mechanism by which they are generated as real-time video warps and maps. The digital Toaster FX include video fly-ins and fly-outs, slats and blinds, mosaics, and many more. The circuit mechanisms by which they are performed on-the-fly is sophisticated, but making and modifying them is easy. Among Toaster developers and programmers, these are known as 'Algo' effects (AlgoFX for short) because they are created by algorithmic methods. First a brief, partial explanation of the Toaster's DVE capabilities and then we'll present a DVE-creation environment with an example.

Video Toaster DVE Theory

The Toaster can continuously digitize, store, 'undigitize' and replay an incoming NTSC video signal. See Figure 1, top portion, and note the digital video frame stores (RAM0, RAM1) and the digital video flow pathway (ADC, FIFO, RAM0/1, DAC0/1). When the Toaster is in 'digital' mode, the two frame stores (which are shown as highlighted buttons on the Switcher screen in digital mode) contain digitized versions of whatever video source is incoming on the MAIN Toaster bus. Except for a very brief switchover delay when the Toaster enters its digital mode, you may not see any difference between the Toaster's output when it is in digital mode and when it is not. The Toaster's digital mode is familiar to almost all Toaster users who've played around with the digital warping transition effects that made the Toaster so popular. The contents of the two framestores are continuously updated and the video which is being digitized, undigitized, and replayed ordinarily appears as any other video source to the Toaster, with the exception that it is very slightly delayed by this process. What goes into the framestores immediately comes out with virtually no change - unless a DVE is taking place.

The digital framestores are simply X,Y arrays of digitzed video values (D2 values). The key to Toaster DVEs is a circuit which can change the order in which the D2 values are played back out of the framestore arrays. If we could rearrange the X,Y arrays fast enough, you can imagine that we might digitize a field of video, turn it upside down (VERY quickly) and then undigitize it and play it out. We'd only have a few milliseconds to completely rearrange the X,Y array in the framestore, though. The Toaster isn't quite that fast. Once a field of video is digitized into the framestore memory (on-the-fly), though, the way it will appear a moment later when it is 'undigitized' depends on the X,Y addressing scheme we apply to the 'undigitizing' circuit. Figure 1 shows the DVE ADDR GENERATOR (the DVE Address Generator), connected to the two framestore memories in the digital video pathway at the top. The Toaster can apply various schemes of X,Y addressing to the framestore's 'undigitzing' process so that the D2 values come out of the framestore memory in a different order than they entered. There are two limitations. Because the Toaster can't do an arbitrary rearrangement of the whole field all at once, it must operate one line at a time. That means the Toaster can only perform real-time horizontal rearrangements. The other limitation is part of the nature of D2 video. It takes four pixels of D2 video to properly reconstruct the color of any one of those four pixels. D2 works in 'quads'. So the Toaster can only rearrange groups of four pixels at a time. Within those two limitations, the Toaster is free to digitally rearrange video on-the-fly as it passes through the framestore memory arrays. Any 'quad' of pixels can be extracted from its original position on a line of video and made to appear at a different location.

Toaster algorithmic DVEs (an Algo FX) feeds the DVE Address Generator according to formulae (algorithms) manipulated in software. You can imagine without much difficulty that we might program a series of horizontal rearrangements of pixel quads to make the video appear to undergo a series of changes which look like it is being squeezed or unsqueezed horizontally. The algorithm for doing that is fairly simple - we can take pixel quads from the edges of the video frame store array

and move them closer to the center. As we progress, we may simply leave out every other quad as the squeeze proceeds to 1/2 of the ordinary picture's width. And so on. If we apply a series of image rearrangements like this fairly quickly we can make a very good (if not quite perfect because of the quad limitation) squeezing effect.

The AlgoFX are a family of DVEs based on fairly simple algorithms for manipulating the DVE Address Generator in real time. Now that you've been introduced to the basic AlgoFX ideas, you can see how blinds, slats, bouncing transitions, mosaics, and so on are all variations on the same theme of image rearrangement in the digital video framestores. As the contents of the frame store memories are replaced every 1/60 of second with a new video field's D2 values, the DVE Address Generator's 'look-up-table' gets updated by the Toaster with a series of address rearrangements for the pixel quads. As long as we can change the addressing scheme for the frame store memories every 1/60 of a second, we can make a series of pixel quad rearrangements appear to move the video smoothly. AlgoFX are made of a series of look-up-tables which can be created in real time. The algorithm for creating a series of look-up-tables which make a square appear to squeeze into a vertical line and disappear is pretty easy to make and to update in real time.

AlgoFX can be spotted by any eagle-eyed Toaster user. True AlgoFX don't cause the Toaster's Switcher display screen to temporarily vanish and shimmer with dim vertical bars or animated shadows. AlgoFX are one kind of Toaster effect which doesn't involve playing Amiga screen graphics to control the Toaster's output on every video pixel. Some Toaster DVEs do show the typical animated dim vertical bars during the effect. Those are a hybrid type of effect in which the DVE Address Generator is operated by an Amiga animation instead of by simple equations. Some of the more complex Digital FX work this way instead - as there is no fast software formula to calculate the series of address look-up-tables in real time. The fast mappings of video onto spheres, etc. are of this hybrid type and aren't true AlgoFX, they are in the hybrid class of Anim/DVEs. These mappings are performed via Amiga animations playing into the DVE Address Generator. True AlgoFX are purely software devices and leave the Toaster display intact. All the DVEs which are digital in nature (which use the DVE Address Generator applied to digitized video in the framestore memories) force the Toaster into its digital mode, though, and clear whatever was in the Toaster's framestores before the effect took place. DVEs always highlight the two framestore buttons on the Switcher display. The hybrid DVEs do, however, reveal the limitation of the Toaster's DVE Address Generator. If you play them slowly, you'll see that they always consist solely of horizontal rearrangements.

All of the other issues related to AlgoFX are based on the operation of the Toaster as explained in the document, NewTek Video Toaster FX Theory (11/28/94). The multiplexer and analog video routing through the two-source video fader is the same as for any group of effects. Notice on the diagram that the AMUX and BMUX both can pass video from the 'undigitizing' circuits (digital to analog converters, DAC0 and DAC1) out to the Fader. That lets the AlgoFX-modified video from one source mix with and fade replace or be removed from another source. AMUX might send video from the DACs to one end of the Fader while BMUX sends a different source to the other end. The Fader's alternation between the two sources can be triggered by any of the inputs to the FCMUX. When video is squeezed by an Algo effect, the DACs may output a value of 0 in those areas where video has been 'removed' so that a hard LUMKEYA value of 1 or 0 could swing the fader completely to the other non-squeezed video source in those areas. That's precisely what you see when video is moved onto or off the Toaster output via an Algo effect. Those areas which have been squeezed or cleared show through (via the Fader, pixel by pixel if necessary) to the other video source.

The AlgoTools Development System

The AFX_CF disk will install a set of development tools you can use to modify and produce new Algo DVEs for the Toaster. The toolkit is quite similar to that presented at the December, 1994 NewTek DevCon for creating anim and color cycling effects. The complete AlgoFX toolkit consists of two drawers, Envir and Make4.0. The Envir drawer contains all the Toaster assembly language include files in the Inc subdirectory (current revisions of eflib.i, tags.i, crouton.i, etc.), the binary files

(PhxAss assembler, date comparison, file merging functions, etc.) in the Bin subdirectory, and the Batch or script files in the Bat subdirectory. For your convenience, I've rewritten the setup and makefile scripts so they will operate in your Amiga system (without having the whole NewTek network file system present). When you install the AFX_CF disk, the AlgoSetup, AlgoMake.DW, and KillAlgo scripts will be placed in your s: directory.

The Make4.0 drawer contains CrUD (Crouton User Data) source files (see NewTek Video Toaster FX Theory, 11/28/94), a generic crouton Icon file, and the Toaster/Effects drawer into which the AlgoMake.DW script will put each finished effect. This toolkit contains all the CrUD source files for currently implemented AlgoFX.

Because this version of the effects environment toolkit doesn't have to work with large anim files, I've modified the AlgoSetup script (which is called when you first execute the AlgoMake.DW script on a Crud file) to move the whole environment to ram: for speed and convenience in prototyping AlgoFX.

Creating an Algo DVE with AlgoMake.DW

As presented in the earlier FX creation tools documentation, the process of making an effect is:

1. Create (or modify an existing) a CrUD source file

It is easiest to simply work with existing CrUD source files and modify them with a text editor. CrUD source files for AlgoFX are highly stereotyped and contain all the necessary elements of a complete effect in a few variables and constants at the end of the CrUD source itself.

2. Execute the AlgoMake.DW script with your selected CrUD file as its argument

Like this:

AlgoMake.DW AlgoCrUDFileName

The AlgoMake script first calls the AlgoSetup script which moves the AlgoTools drawer into your ram disk and creates numerous assigns to the existing constellation of directories in the AlgoTools drawer. For example, CSRC becomes an assign name for the CrUD source file drawer, AlgoTools/Make4.0/Csrc/CrUD. If you'd like to study and review it, the AlgoMake.DW script is virtually identical to the SingleMake script which was annotated and printed in the NewTek Video Toaster FX Theory (11/28/94) document. If you examine the contents of Make4.0 and Envir, you'll find they closely resemble those supplied as part of the FXTools system (which included anim, shadow, and color cycling effects) at DevCon94 and described in the document FX Theory document (11/28/94).

Creating a new Algo effect is as simple as the two steps above, and as complicated as manipulating the data items in an Algo CrUD file. There are a number of classes of AlgoFX and literally dozens of CrUD files included in this package, so we'll examine one of them and use it as an example of how to modify one effect to create a new one or two. Additional experimentation with the other myriad of CrUD files is left as an exercise for the developer.

An Example Algo DVE - Blinds 3 Expand

The following is a complete printout of the CrUD source file for the Blinds 3 Expand Algo effect, with a little added commentary. If necessary, please review the NewTek Video Toaster FX Theory document (11/28/95) to recount the basic structure of a CrUD file. The CrUD file for an Algo effect of this type (blinds) consists of only a few tags (which are defined in the tags.i file included on the first line, and which you can find in the AlgoTools/envir/inc drawer). The tags which you should note immediately below are the tag macros CrUD_START and CrUD_AlgoFX which define this type of effect CrUD source as an Algo effect (the definitions for these tag macros are in tags.i). The other interesting tag is the last one, TABLE_Equations which form the real essence of an Algo effect CrUD - they point to the collection of constants at the end of the CrUD file itself which are the parameters the Toaster's internal Algorithmic Effect generator interpret to manipulate the DVE Address Generator. The last constant in the TABLE_Equations set is the number, 3. Note the comment here indicates the this number denotes the number of rectangles (blinds) for this effect.

```
    include "tags.i"

    CrUD_START    CrUD_AlgoFX,CrUD_4_0,CrUD_BWIcon

    TAG_FCountMode      1
    TAG_VariableFCount   0
    TAG_SlowFCount      60
    TAG_MedFCount       45
    TAG_FastFCount      30
    TAG_AlgoFXtype        ALGT_StdDigitalFX
    TAG_ButtonELHlogic    AFXT_Logic_TDEon

    TABLE_Equations
10$     dc.l  20$-10$               ;always points to Time Variables
        dc.l  DEF_REVERSE           ;modes
        dc.w  3                     ;num rectangles horizontally
20$:
```

Guess what happens if we change the 3 to a 2, above, with no other changes to this CrUD source file....

The following comments can guide you when you explore the meanings of some of the tag and constant defnitions in tags.i, eflib.i, rect.i, crouton.i, etc. Note the '03=linear ...' comment which informs us that the four numbers following a 'LINEAR' tag constant later in this source file will indicate the distance and speed variables which move the rectangles at a constant speed. The '04=accelerated ...' comment informs us similarly about the meanings of the five numbers which accompany a 'ACCEL' tag. This effect does indeed use LINEAR tags to specify motion of the three rectangles which make up this effect (see below).

```
;-----------------------------------------
;00 = end of table
;01 = ignore
;02 = constant        WORDvalue

;all 03,04,05 parameters are binary reals (LONGS)
;03 = linear          initDistance, speed, minDistance, maxDistance
; mind <= sp*t+id <= maxd

;04 = accelerated      initVelocity, accel, minSpeed, maxSpeed, initDistance
; (minsp <= ac*t+iv <= maxsp)*t + id
```

Harmonic motions are bouncing motions in a sinusoidal manner. You'll see them used in some of the effects which move video onto or off the screen in a jiggling manner.

```
;05 = harmonic        initFreq, deltaFreq, minFreq, maxFreq, initPhase,
;                     iAmp, deltaAmp, minAmp, maxAmp
; (minam <= da*t+ia <= maxam)*t * SIN[(minfr <= df*t+if <= maxfr)*t + ip]+os
```

Now we find a fixed-structure table of values which designate the horizontal (X) behavior (scaling) of each of the three rectangles. Note that this type of effect can have only a single Y behavior which governs all of the rectangles the same way, but X behavior can be manipulated differently for each rectangle. That's handy, as we really do wish to (at least) place each of the rectangles at different horizontal positions. The ET_DVE1X tells the Toaster's AlgoFX software that this is the start of the position and motion table of values for rectangle #1. Later you'll find ET_DVE2X and ET_DVE3X which denote similar table start declarations for the other two rectangles in this effect. Each of the rectangle position and motion tables is in a fixed form and the comments you'll find tell you the meaning of each item. TVT_CONST means the following value is a constant for the whole effect. TVT_LINEAR denotes a set of four numbers follows which dictate a linear (smooth) motion, etc. SXMIN means Source X Minimum, the leftmost position from which source video will arise (0). SXSIZE is the Source X Size in 'quads', namely a full screen width (185 quads). The values for these and most of the other constants (DXMIN = Destination X Minimum, SXMAX, etc.) are found in croutons.i.

```
;XSCALE 1
     dc.w ET_DVE1X1

;borderUpper
     dc.w TVT_CONST
     dc.w $0000

;borderLower
     dc.w TVT_CONST
     dc.w $0000
```

We want the center position of the leftmost rectangle's source video, which will occupy one third of the screen, to originate 1/6 of the way from the left boundary of the source video.

```
;position
     dc.w TVT_CONST
     dc.w SXSIZE/6
```

The minimum width of the source video is 2 quads.

```
;srcMin
     dc.w TVT_CONST
     dc.w SXMIN
```

The maximum width of the source video for this rectangle is 1/3 of the source's total width.

```
;srcMax
     dc.w TVT_CONST
     dc.w SXMIN+SXSIZE/3
```

The center of the region of source video from which this rectangle is drawn is 1/6 of the way from the left of the source video boundary.

```
;srcAxis
      dc.w TVT_CONST
      dc.w SXMIN+SXSIZE/6
```

You can see that the destination positions and axes are arranged to correspond to the positions at which we want the source video to be 'painted' into the destination frame.

```
;destMin
      dc.w TVT_CONST
      dc.w DXMIN

;destMax
      dc.w TVT_CONST
      dc.w DXMAX

;destAxis
      dc.w TVT_CONST
      dc.w DXAXISMIN
```

The size of the rectangle will increase from thin to thick (see comments earlier in this file which summarize the meaning of the four numbers which follow this tag).

```
;effectSize
      dc.w TVT_LINEAR
      dc.l (SXSIZE/3+1)<<16,-(SXSIZE/3+1)<<16,$00000000,MAXLIMIT

;effectColor
      dc.w TVT_CONST
      dc.w $0000

;borderColor
      dc.w TVT_CONST
      dc.w IGNORE

;backgroundColor
      dc.w TVT_CONST
      dc.w BGCOLORX

;overscanColor
      dc.w TVT_CONST
      dc.w IGNORE
```

This denotes the end of the position/motion table for this first rectangle.

```
      dc.w TVT_END
*------
```

Here is the start of the position/motion table of values for the second rectangle.

```
;XSCALE 2
      dc.w ET_DVE1X2

;borderUpper
      dc.w TVT_CONST
      dc.w $0000
```

```
;borderLower
    dc.w TVT_CONST
    dc.w $0000
```

The position of this rectangle is in the center (x-size divided by 2).

```
;position
    dc.w TVT_CONST
    dc.w SXSIZE/2
```

The minimum x position of source video for this rectangle starts 1/3 of the way from the left of the source (the first rectangle starts all the way at the source video's left edge).

```
;srcMin
    dc.w TVT_CONST
    dc.w SXMIN+SXSIZE/3+1
```

The maximum x position for the source video of this rectangle is 2/3 from the source's left edge. This rectangle will bring in the middle 1/3 of source video onto the screen. Compare this value to that of the previous rectangle which brings in the leftmost 1/3 of source video.

```
;srcMax
    dc.w TVT_CONST
    dc.w SXMIN+(SXSIZE*2)/3
```

The axis for the source video of this rectangle is in the center of the source video.

```
;srcAxis
    dc.w TVT_CONST
    dc.w SXMIN+SXSIZE/2

;destMin
    dc.w TVT_CONST
    dc.w DXMIN

;destMax
    dc.w TVT_CONST
    dc.w DXMAX

;destAxis
    dc.w TVT_CONST
    dc.w DXAXISMIN
```

Again we prescribe a linear smooth motion for this rectangle but it must start at a different position (InitDistance) than the previous rectangle. This one will start 1/3 of the way across the destination.

```
;effectSize
    dc.w TVT_LINEAR
        dc.l ((SXSIZE*2)/3-(SXSIZE/3))<<16,-((SXSIZE*2)/3-
        (SXSIZE/3))<<16,$00000000,MAXLIMIT

;effectColor
    dc.w TVT_CONST
    dc.w $0000
```

```
;borderColor
      dc.w TVT_CONST
      dc.w IGNORE

;backgroundColor
      dc.w TVT_CONST
      dc.w IGNORE

;overscanColor
      dc.w TVT_CONST
      dc.w IGNORE
```

Here's the end of this rectangle's X data table.

```
      dc.w TVT_END
*------
```

Here's the start of the rightmost (third) rectangle's X data.

```
;XSCALE 3
      dc.w ET_DVE1X3

;borderUpper
      dc.w TVT_CONST
      dc.w $0000

;borderLower
      dc.w TVT_CONST
      dc.w $0000
```

This is positioned 5/6 of the way from the left.

```
;position
      dc.w TVT_CONST
      dc.w (SXSIZE*5)/6

;srcMin
      dc.w TVT_CONST
      dc.w SXMIN+(SXSIZE*2)/3+1

;srcMax
      dc.w TVT_CONST
      dc.w SXMAX
```

The axis corresponds to the position, SXMIN (nearly zero) plus 5/6 of the SXSIZE.

```
;srcAxis
      dc.w TVT_CONST
      dc.w SXMIN+(SXSIZE*5)/6

;destMin
      dc.w TVT_CONST
      dc.w DXMIN
```

The rectangle extends to the bottom of the screen.

```
;destMax
      dc.w TVT_CONST
      dc.w DXMAX

;destAxis
      dc.w TVT_CONST
      dc.w DXAXISMIN
```

Again we see a linear motion for this rectangle, but with a different InitDistance. You could change this to:

```
;TVT_CONST
;dc.w SXSIZE/3
```

to try an interesting variation in which this rectangle would come onto the screen at a fixed size and stay that way throughout the effect instead of varying linearly.

```
;effectSize
      dc.w TVT_LINEAR
            dc.l (SXMAX-SXMIN-(SXSIZE*2)/3)<<16,-(SXMAX-SXMIN-
              (SXSIZE*2)/3)<<16,$00000000,MAXLIMIT

;effectColor
      dc.w TVT_CONST
      dc.w $0000

;borderColor
      dc.w TVT_CONST
      dc.w IGNORE

;backgroundColor
      dc.w TVT_CONST
      dc.w IGNORE

;overscanColor
      dc.w TVT_CONST
      dc.w IGNORE
```

Here's the end of the X data table for rectangle #3.

```
      dc.w TVT_END
```

Here's the start of the Y data table for all three rectangles.

```
;---------------
;YSCALE
      dc.w ET_DVE1Y

;borderUpper
      dc.w TVT_CONST
      dc.w $0000
```

```
;borderLower
      dc.w TVT_CONST
      dc.w $0000

;position
      dc.w TVT_CONST
      dc.w   0
```

You could try changing this to a larger value and see the source video start further down the source's vertical span, instead of from its top.

```
;srcMin
      dc.w TVT_CONST
      dc.w SYMIN
```

You could change this to a smaller value and the source video's vertical span would not extend all the way to the bottom of the source video's vertical extent. For example, SYMAX/2 would restrict the amount of source video to 1/2 of the source's vertical span. As the rectangles come on screen, you wouldn't see them bringing in all of the source, only its upper half.

```
;srcMax
      dc.w TVT_CONST
      dc.w SYMAX

;srcAxis
      dc.w TVT_CONST
      dc.w SYAXIS

;destMin
      dc.w TVT_CONST
      dc.w DYMIN

;destMax
      dc.w TVT_CONST
      dc.w DYMAX

;destAxis
      dc.w TVT_CONST
      dc.w DYAXIS
```

This tells the Toaster to bring in all three rectangles at the full size of the video's Y dimension.

```
;effectSize
      dc.w TVT_CONST
      dc.w SYSIZE

;This will tell the Toaster to bring the rectangles in, changing their sizes linearly from small to the full
;display height during the course of the effect instead of full-height from the start - this makes an
;interesting and useful variation of the effect.

;  substitute these instead to make a nifty change!
;       dc.w TVT_LINEAR
;       dc.l (SYSIZE)<<16,-(SYSIZE)<<16,$00000000,MAXLIMIT
```

```
;effectColor
    dc.w TVT_CONST
    dc.w $0000

;borderColor
    dc.w TVT_CONST
    dc.w IGNORE

;backgroundColor
    dc.w TVT_CONST
    dc.w BGCOLORY

;overscanColor
    dc.w TVT_CONST
    dc.w IGNORE

;Here's the end of the Y position/motion table of values for the rectangles.

    dc.w TVT_END
```

This tag tells the Toaster's effects generation software to 'run' its 'equations' based on the parameters from the tables included above.

```
    dc.w ET_RUN
```

This is the end of all the tables for this effect.

```
    TABLE_END
```

This is the CrUD_END macro tag (from tags.i) which tells the assembler to finish up here.

```
    CrUD_END
```

The effect we've just examined, "Blinds 3 Expand" flies three vertical rectangles of source video onto the screen. Each rectangle brings in 1/3 of the source onto the destination frame, is positioned at the appropriate 1/3 of the screen, and increases in size to reveal a wider and wider vertical swath of 1/3 of the source video until it fills the screen.

We'll use this CrUD file as a template to experiment with a couple of potentially interesting variations. The best way for you to gain experience with creating new AlgoFX is to intelligently modify one or two parameters of this kind of CrUD file and use it immediatly with your Toaster to view the consequences. Sometimes you can easily predict the results you'll obtain by making a change (and you'll gain experience at that with time). Sometimes you'll find that a modification doesn't have any effect at all - probably due to exceeding a limit which is enforced by the Toaster's internal software AlgoFX handler. I've found no effect of modifying the border color tags at all - maybe you can. Sometimes you'll find completely serendipitous results arise from your experiments. When you've created something new and tasty, save the modified CrUD file and keep the effect. The environment we've provided, AlgoTools, is a playground for your imagination and creativity.

Happy AlgoMaking.

DW
10/26/95

```
*************************************************************************
* rect.i
*
* Copyright (c)1992 NewTek, Inc.
* Confidental and Proprietary. All rights reserved.
*
* $Id: rect.i,v 2.2 93/12/02 20:48:32 Kell Exp $
*
* $Log:  rect.i,v $
*Revision 2.2  93/12/02  20:48:32  Kell
*Now has structures for dynamic Fader/LumKey/TBarTime/BMposition.
*New structures for new types of motion for 4.0
*
*Revision 2.1  93/11/06  08:45:51  Kell
*Fixed double defined label (structure name)
*
*Revision 2.0  92/05/18  21:23:54  Hartford
**** empty log message ***
*
*************************************************************************
   IFND  RECT_I
RECT_I   SET   1

* if >0 do ELH
* if 3 = first
* if 5 = last

FIELDDURING EQU   0  ;transformed to DOELH or NOELH via ProcessDuring()
FIELDNOELH  EQU   0  ;during stage, no elh
FIELDDOELH  EQU   1  ;during stage, do elh
FIELDFIRST  EQU   3  ;1st of stage, do elh
FIELDLAST   EQU   5  ;may do take, do elh
FIELDREDO   EQU   7  ;redraw as LAST, with elh, but logic of FIRST

DIGOFFSET   SET   -3 ;relation of source to dest quad position
QUAFBYTE SET   $ff   ;transparent digital quad

*********************************************************
   STRUCTURE   ScaleData,0

;I don't see much use in border that has a non-constant thickness, but
;maybe this could be used in advanced drop shadows someday.
     WORD  SD_borderUpper  ;thickness of border, in destin. resolution
     WORD  SD_borderLower  ;thickness of border, in destin. resolution
     WORD  SD_position  ;position of srcaxis with respect to destaxis,
            ;in destination resolution

;source cropping and scaling axis placement.
;Source cropping on Y sources is a bit weird because of the EVENODD
;rule. So you may need to "undercut" to avoid Y scaling from falling off edge.
     WORD   SD_srcMin   ;minimum address to ever use from source
     WORD   SD_srcMax   ;maximum address to ever use from source
     WORD   SD_srcAxis  ;axis to scale from

;NOTE!!!!! At the current time, maybe srcAxis has to lie between srcMin and
;srcMax inclusive ?????
;It crashed when I tried Min/Max/Axis = SXAXIS/SXMAX/SXMIN

;Y scaling will require two srcAxis, an even and an odd one.  So this
;SD_srcAxis will be one of them, and SD_srcAxis+1 will be the other!

;destination cropping, defines useful underscan area
     WORD   SD_destMin  ;minimum address to ever use in destination
     WORD   SD_destMax  ;maximum address to ever use in destination
     WORD   SD_destAxis ;display coordinate origin.

     WORD  SD_effectSize   ;width or height of effect, in dest resolution
;effectSize is place here because it has to be after srcMax & srcMin have
;been defined. Because srcSize may effect effectSize.

;the effect&bordercolors are ignored for DVE Input Effects, i.e. Trails.
;but backgroundColor is looked at when horizontal scaling.
;if vertical scaling, give least of two source lines.
     WORD  SD_effectColor     ;>0 color of effect, -1=skip effect, 0=doit
     WORD  SD_borderColor     ;color of border, -1=skip border
```

```
     WORD  SD_backgroundColor ;color of background, -1=skip bg
;currently overscancolor is not utilized
          WORD  SD_overscanColor   ;color of overscan area, -1=skip os

;the Scale routine will determine the remaining elements in this structure.
     WORD  SD_rectMin   ;minimum/maximum destination addresses that
     WORD   SD_rectMax  ;have an effect element, on or off screen.

     WORD  SD_effectMin    ;min/max on screen destination address that
     WORD  SD_effectMax ;has a scaled effect.  Is not altered by borders.
            ;=-1 if no scaled effect on screen.
            ;Useful for pixelizing all but borders.

     WORD  SD_totalMin     ;min/max on screen destination address that
     WORD  SD_totalMax  ;has an effect or borders.  May be altered by
            ;border routine. =-1 if nothing on screen.

;parameters to scale from destInit and go upward
     WORD  SD_destInit  ;address of initial destination address
     LONG  SD_srcInit   ;binary real, source initial value
     LONG  SD_srcInc ;binary real, source incremental value
     WORD  SD_countUp   ;# of points to plot, destInc = 1

;parameters to scale from destInit and go downward
;use neg.l srcInc for downward source incremental value
     WORD  SD_countDown ;# of points to plot, destInc = -1

   LABEL SD_SIZEOF         ;size of this structure

************************************************************************
BYTE_SIZEOF SET 1
WORD_SIZEOF SET 2
LONG_SIZEOF SET 4

* values to use so min/max motion parameters are ignored
MINABSREAL  SET 0    ;use this if you have no min limit
MAXABSREAL  SET $7fffffff  ;use this if you have no max limit

* All of the variables in the Accelerate/Harmonic structures are
* binary Reals.
* So that cropping of the source does not cause scaling in the final
* effectSize, changes in crop may tweek sp, by adding
* (srcSizeNew-srcSizeOld) to sp.

   STRUCTURE Linear,0   ; mind <= sp*t+id <= maxd
     LONG   LM_initDistance
     LONG   LM_speed
     LONG   LM_minDistance ;value >= 0
     LONG   LM_maxDistance  ;value >= 0
   LABEL LM_SIZEOF   ;size of this structure

*---------------------------------------
* So that cropping of the source does not cause scaling in the final
* effectSize, changes in crop may tweek iv, by adding
* (srcSizeNew-srcSizeOld) to iv.

* (minsp <= ac*t+iv <= maxsp)*t + id
   STRUCTURE Accelerate,0
     STRUCT AC_accel,LM_SIZEOF
            ;LONG AC_initVelocity
            ;LONG AC_acceleration
            ;LONG AC_minSpeed ;min abs(Velocity)
            ;LONG AC_maxSpeed ;max abs(velocity)
     LONG   AC_initDistance
   LABEL AC_SIZEOF   ;size of this structure

*---------------------------------------
* So that cropping of the source does not cause scaling in the final
* effectSize, changes in crop may tweek ia, by adding
* (srcSizeNew-srcSizeOld) to ia.

*  (minam <= da*t+ia <= maxam) * SIN[(minfr <= df*t+if <= maxfr)*t + ip] + os
   STRUCTURE   Harmonic,0
     STRUCT HM_frequency,LM_SIZEOF
            ;LONG HM_initFreq
```

```
            ;LONG HM_deltaFreq
            ;LONG HM_minFreq  ;min abs(Frequency)
            ;LONG HM_maxFreq  ;max abs(Frequency)
    LONG    HM_initPhase

    STRUCT HM_amplitude,LM_SIZEOF
            ;LONG HM_initAmp
            ;LONG HM_deltaAmp
            ;LONG HM_minAmp   ;min abs(Amplitude)
            ;LONG HM_maxAmp   ;max abs(Amplitude)
    LONG  HM_initOffset
  LABEL HM_SIZEOF          ;size of this structure

*-----------------------------------------
  STRUCTURE DelayedLinear,0  ; mind <= sp*t+id <= maxd
    LONG    DLM_initDistance
    LONG    DLM_speed
    LONG    DLM_minDistance   ;value >= 0
    LONG    DLM_maxDistance  ;value >= 0
    WORD    DLM_timeStart  ;value >= 0
    WORD  DLM_timeEnd  ;value >0, else = $10000
  LABEL DLM_SIZEOF  ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaLinearData,0
    LONG    DLD_Delta
  LABEL DLD_SIZEOF  ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaAccelData,0
    LONG    DAD_Delta
  LABEL DAD_SIZEOF  ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaDeaccelData,0
    LONG    DDD_Delta
  LABEL DDD_SIZEOF  ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaSCurveData,0
    LONG    DSD_Delta
  LABEL DSD_SIZEOF  ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaBigAccelData,0
    LONG    DBAD_Delta
  LABEL DBAD_SIZEOF ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaBigDeaccelData,0
    LONG    DBDD_Delta
  LABEL DBDD_SIZEOF ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaRandomData,0
    LONG    DRD_Delta
  LABEL DRD_SIZEOF  ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaConstantData,0
    LONG    DCND_Delta
  LABEL DCND_SIZEOF ;size of this structure

*-----------------------------------------
  STRUCTURE DeltaCyclicData,0
   LONG DCD_initAmp
   LONG DCD_deltaAmp
   LONG DCD_initPhase
   LONG DCD_deltaPhase
   LONG DCD_deltaFreq
  LABEL DCD_SIZEOF  ;size of this structure

*-----------------------------------------
* If TV_function is <0 then ignore.  If TV_function = NULL then end of table
* All TV_functions will be called with;
```

```
 * a0->data (ie Accelerate/Harmonic structures),
 * a1->results (ie ScaleData structure field), d0=Time
    STRUCTURE   TimeVariablesStruct,0
      APTR   TV_function ;->a motion/size function
      APTR   TV_data     ;->data used by the function, NULL if none
      APTR   TV_results  ;->place to put results
    LABEL  TV_SIZEOF  ;size of this structure

 *---------------------------------------
 *
 * The pixelize routines are ment to pixelize the buffX or buffY buffers.

    STRUCTURE   PixelizeData,0
 ;destination cropping, defines useful underscan area
      WORD   PD_destMin  ;minimum address to ever use in destination
      WORD   PD_destMax  ;maximum address to ever use in destination
      WORD   PD_destAxis ;display coordinate origin.

      WORD  PD_blockSize ;width or height of effect, in dest resolution
    LABEL PD_SIZEOF

 ************************************************
    STRUCTURE   Wipe1Data,0

      WORD  W1D_position ;position of srcaxis with respect to destaxis,
            ;in destination resolution
      WORD   W1D_srcAxis ;axis to wipe from, =center of wiped rectangle
            ;based on a 10000 unit wide rectangle
            ;0=use left edge of rect, 9999 use right, 5000 use center
 ;destination cropping, defines useful underscan area
      WORD   W1D_destMin ;minimum address to ever use in destination
      WORD   W1D_destMax ;maximum address to ever use in destination
      WORD   W1D_destAxis   ;display coordinate origin.

      WORD  W1D_effectSize  ;width or height of effect, in dest resolution

      WORD  W1D_effectColor ;1=> set effect, clear BG
           WORD  W1D_overscanColor   ;color of overscan area, -1=skip os

 ;the wipe routine will determine the remaining elements in this structure.
      WORD  W1D_rectMin  ;minimum/maximum destination addresses that
      WORD   W1D_rectMax ;have an effect element, on or off screen.

      WORD  W1D_effectMin    ;min/max on screen destination address that
      WORD  W1D_effectMax   ;has a scaled effect.
            ;=-1 if no scaled effect on screen.
    LABEL W1D_SIZEOF        ;size of this structure

 ************************************************
    STRUCTURE   EffectsLUT,0
       APTR  ELUT_variables
       APTR  ELUT_equations
    LABEL ELUT_SIZEOF

 ************************************************
 * Toaster Register Time Variables, dynamic w/o matt
    STRUCTURE   ToasterELH1,0
       WORD  TELH1_CD    ;control da
       WORD  TELH1_LKA   ;lumkey A
       WORD  TELH1_LKB   ;lumkey B
    LABEL TELH1_SIZEOF

 * Toaster Register Time Variables, non-dynamic w/o matt
 * Other static bits could go in here, but remember to change NUMTV_TELH2
    STRUCTURE   ToasterELH2,0
       WORD  TELH2_AM
       WORD  TELH2_BM
       WORD  TELH2_IS
       WORD  TELH2_PV
       WORD  TELH2_LK
       WORD  TELH2_CDS
    LABEL TELH2_SIZEOF

 * Toaster Register Time Variables, matt & (autocal?)
    STRUCTURE   ToasterELH3,0
```

```
      WORD  TELH3_MA     ;matt reg A
      WORD  TELH3_MB     ;matt reg B
      WORD  TELH3_MP     ;matt phase
   LABEL TELH3_SIZEOF

************************************************
* used by 4.0 ANIM/ILBM Effects
   STRUCTURE   FaderData,0
      UWORD FDRD_CD      ;0-255
   LABEL FDRD_SIZEOF

*--------------------
   STRUCTURE   LumKeyData,0
      UWORD LKAD_LKA ;0-255
   LABEL LKAD_SIZEOF

*--------------------
   STRUCTURE   TBarTimeData,0
      UWORD TBTD_Time    ;0-$ffff
   LABEL TBTD_SIZEOF

*--------------------
   STRUCTURE   BMpositionData,0
      WORD  BMPD_Xaxis  ;source, lines up with Destination X axis when deltaX = 0
      WORD  BMPD_Yaxis  ;source, lines up with Destination Y axis when deltaY = 0
      WORD  BMPD_deltaX
      WORD  BMPD_deltaY
   LABEL BMPD_SIZEOF

************************************************
* This structure is always at the start of all crouton DVE data
   STRUCTURE   DVEEffects,0
      APTR  DVEE_TimeVars
;;    APTR  DVEE_EntrySetUp  ;entry toaster register config.
;;    APTR  DVEE_ExitSetUp  ;toaster register config on exit
      LONG  DVEE_Modes1
      WORD  DVEE_NumHor
   LABEL DVEE_SIZEOF

* DVEE_Modes1

DEB_SHOWFACE   SET   0  ;leave interface up while doing effect
DEB_DRAGTBAR   SET   1  ;show tbar dragging
DEB_REVERSE SET   2  ;reverse the direction of the effect
DEB_MIDFLOP    SET   3  ;when TBar is halfway down, switch sources and reverse
DEB_DOELH1  SET   4  ;frames do SendELH2Toaster (w/o MATT/dynamic)
DEB_DOELH2  SET   5  ;frames do SendELH2Toaster (w/o MATT/non-dynamic)
DEB_DOELH3  SET   6  ;frames do SendELH2Toaster (MATT & autocal)
DEB_GEOONINPUT SET   7  ;set if effects on writing, else on playback
DEB_STAMPWAIT  SET   8  ;used with Writing effects, to hold TBar 8 fields
DEB_TILEX   SET   9  ;duplicate geometry to fill buffer
DEB_TILEY   SET   10
DEB_MOSAICX SET   11 ;mosaic will be applied after DVE is calculated
DEB_MOSAICY SET   12

*-----------
DEF_SHOWFACE   SET   (1<<DEB_SHOWFACE)
DEF_DRAGTBAR   SET   (1<<DEB_DRAGTBAR)
DEF_REVERSE SET   (1<<DEB_REVERSE)
DEF_MIDFLOP    SET   (1<<DEB_MIDFLOP)
DEF_DOELH1  SET   (1<<DEB_DOELH1)      ; w/o MATT, dynamic
DEF_DOELH2  SET   (1<<DEB_DOELH2)      ; w/o MATT, non-dynamic
DEF_DOELH3  SET   (1<<DEB_DOELH3)      ; MATT & AutoCal
DEF_GEOONINPUT SET   (1<<DEB_GEOONINPUT)
DEF_STAMPWAIT  SET   (1<<DEB_STAMPWAIT)
DEF_TILEX   SET   (1<<DEB_TILEX)
DEF_TILEY   SET   (1<<DEB_TILEY)
DEF_MOSAICX SET   (1<<DEB_MOSAICX)
DEF_MOSAICY SET   (1<<DEB_MOSAICY)
************************************************
* This structure is always at the start of all crouton Wipe data
   STRUCTURE   WipeEffects,0
      APTR  WPE_TimeVars
;;    APTR  WPE_EntrySetUp  ;entry toaster register config.
;;    APTR  WPE_ExitSetUp   ;toaster register config on exit
```

```
      LONG  WPE_Modes1
      WORD  WPE_NumHor
   LABEL WPE_SIZEOF

* WPE_Modes1
WEB_SHOWFACE   SET   0  ;leave interface up while doing effect
WEB_DRAGTBAR   SET   1  ;show tbar dragging
WEB_REVERSE SET   2  ;reverse the direction of the effect
WEB_MIDFLOP SET   3  ;when TBar is halfway down, switch sources and reverse
WEB_DOELH1  SET   4  ;frames do SendELH2Toaster (w/o MATT/dynamic)
WEB_DOELH2  SET   5  ;frames do SendELH2Toaster (w/ MATT/non-dynamic)
WEB_DOELH3  SET   6  ;frames do SendELH2Toaster (MATT & autocal)
WEB_CROSS   SET   7  ;to cross wipe

*-----------
WEF_SHOWFACE   SET   (1<<WEB_SHOWFACE)
WEF_DRAGTBAR   SET   (1<<WEB_DRAGTBAR)
WEF_REVERSE SET   (1<<WEB_REVERSE)
WEF_MIDFLOP SET   (1<<WEB_MIDFLOP)
WEF_DOELH1  SET   (1<<WEB_DOELH1)      ; w/o MATT, dynamic
WEF_DOELH2  SET   (1<<WEB_DOELH2)      ; w/o MATT, non-dynamic
WEF_DOELH3  SET   (1<<WEB_DOELH3)      ; MATT & AutoCal
WEF_CROSS   SET   (1<<WEB_CROSS)

************************************************
* This structure is always at the start of all crouton Switcher data
   STRUCTURE   SwitcherEffects,0
      APTR  SWE_TimeVars
;;    APTR  SWE_EntrySetUp  ;entry toaster register config.
;;    APTR  SWE_ExitSetUp   ;toaster register config on exit
      LONG  SWE_Modes1
   LABEL SWE_SIZEOF

* SWE_Modes1
SEB_SHOWFACE   SET   0  ;leave interface up while doing effect
SEB_DRAGTBAR   SET   1  ;show tbar dragging
SEB_REVERSE SET   2  ;reverse the direction of the effect
SEB_MIDFLOP SET   3  ;when TBar is halfway down, switch sources and reverse
SEB_DOELH1  SET   4  ;frames do SendELH2Toaster (w/o MATT/dynamic)
SEB_DOELH2  SET   5  ;frames do SendELH2Toaster (w/ MATT/non-dynamic)
SEB_DOELH3  SET   6  ;frames do SendELH2Toaster (MATT & autocal)

*-----------
SEF_SHOWFACE   SET   (1<<SEB_SHOWFACE)
SEF_DRAGTBAR   SET   (1<<SEB_DRAGTBAR)
SEF_REVERSE SET   (1<<SEB_REVERSE)
SEF_MIDFLOP SET   (1<<SEB_MIDFLOP)
SEF_DOELH1  SET   (1<<SEB_DOELH1)      ; w/o MATT, dynamic
SEF_DOELH2  SET   (1<<SEB_DOELH2)      ; w/o MATT, non-dynamic
SEF_DOELH3  SET   (1<<SEB_DOELH3)      ; MATT & AutoCal

************************************************
   ENDC  ;RECT_I
```

```
*****************************************************************
* croutons.i
*
* Copyright (c)1992 NewTek, Inc.
* Confidental and Proprietary. All rights reserved.
*
* $Id: croutons.i,v 2.0 92/05/18 21:20:41 Hartford Exp $
*
* $Log:  croutons.i,v $
*Revision 2.0  92/05/18  21:20:41  Hartford
**** empty log message ***
*
*****************************************************************
   IFND  CROUTONS_I
CROUTONS_I  SET   1

*****************************************************************
* Constants used by effects croutons.
*
* By SKell, NewTek  Aug 1990
*****************************************************************

DIGOFFSET   SET -3
DIGOFFSETDTW   SET 1

ZERO     SET 0 ;will always be 0

QUADSWIDE   SET 184
PIXELSWIDE  SET (QUADSWIDE<<2)

MAXLIMIT    SET $7fffffff  ;used by generation equations
IGNORE      SET -1      ;used by overscanColor, etc.

HORSAFETY   SET   0  ;3 quads SAFETY margine from TEK ??
VERTSAFETY  SET   0  ;2 field lines margine from TEK ??

***** STANDARD TDE ************************************************************

*----- TDE HORIZONTAL SOURCE constants for non-trail TDE rectangle
* Use these if the rectangle has no border, includes zipper
SXMIN    SET (5+HORSAFETY)
SXMAX    SET (189-HORSAFETY)
SXAXIS      SET ((SXMAX+SXMIN)>>1)
SXSIZE      SET (SXMAX+1-SXMIN)

* Use these if the rectangle has a border
SXMINBORDER SET SXMIN   ;no diff
SXMAXBORDER SET 182  ;borders don't extend all the way across (0-732).
SXAXISBORDER   SET ((SXMAXBORDER+SXMINBORDER)>>1)
SXSIZEBORDER   SET (SXMAXBORDER+1-SXMINBORDER)

* Use these if the rectangle has a zipper
SXMINZIPPER SET SXMIN   ;no diff
SXMAXZIPPER SET 185     ;186-189.5 = ZIPPER
SXAXISZIPPER   SET ((SXMAXZIPPER+SXMINZIPPER)>>1)
SXSIZEZIPPER   SET (SXMAXZIPPER+1-SXMINZIPPER)

* Use these if the rectangle is in PAIR units
SXMINPAIR   SET SXMIN
SXMAXPAIR   SET SXMAX
SXAXISPAIR  SET ((SXMAXPAIR+SXMINPAIR)>>1)
SXSIZEPAIR  SET (SXMAXPAIR+1-SXMINPAIR)

* Use these if the rectangle is in PAIR units, with border
SXMINPAIRBORD  SET SXMINBORDER
SXMAXPAIRBORD  SET SXMAXBORDER
SXAXISPAIRBORD    SET ((SXMAXPAIRBORD+SXMINPAIRBORD)>>1)
SXSIZEPAIRBORD SET (SXMAXPAIRBORD+1-SXMINPAIRBORD)

*----- TDE HORIZONTAL DESTINATION constants for non-trail TDE rectangle
* Use these if the rectangle has no border, includes zipper
DXMIN    SET ZERO
DXMAX    SET (QUADSWIDE-1)
DXAXISMIN   SET (SXMIN+DIGOFFSET)   ;left most axis
DXAXIS      SET (SXAXIS+DIGOFFSET)  ;axis in center of display
```

```
DXAXISMAX   SET (SXMAX+DIGOFFSET)   ;right most axis
DXSIZE      SET (DXMAX+1-DXMIN)
DXSHALF     SET (DXSIZE>>1)      ;one half the screen size

* Use these if the rectangle has a border
DXMINBORDER SET DXMIN   ;no diff
DXMAXBORDER SET DXMAX   ;no diff
DXAXISMINBORDER   SET (SXMINBORDER+DIGOFFSET)   ;left most axis
DXAXISBORDER   SET (SXAXISBORDER+DIGOFFSET)  ;axis in center of display
DXAXISMAXBORDER   SET (SXMAXBORDER+DIGOFFSET)   ;right most axis
DXSIZEBORDER   SET (DXMAXBORDER+1-DXMINBORDER)
DXSHALFBORDER  SET (DXSIZEBORDER>>1)      ;one half the screen size

* Use these if the rectangle has a zipper
DXMINZIPPER SET DXMIN   ;no diff
DXMAXZIPPER SET DXMAX   ;no diff
DXAXISMINZIPPER   SET (SXMINZIPPER+DIGOFFSET)   ;left most axis
DXAXISZIPPER   SET (SXAXISZIPPER+DIGOFFSET)  ;axis in center of display
DXAXISMAXZIPPER   SET (SXMAXZIPPER+DIGOFFSET)   ;right most axis
DXSIZEZIPPER   SET (DXMAXZIPPER+1-DXMINZIPPER)
DXSHALFZIPPER  SET (DXSIZEZIPPER>>1)      ;one half the screen size

* Use these if the rectangle is in PAIR units
DXMINPAIR   SET (((DXMIN*2)+3)&~3)  ;round up to nearest oct
DXMAXPAIR   SET ((DXMAX*2)&~3)   ;round down to nearest oct
DXAXISMINPAIR  SET ((SXMINPAIR+DIGOFFSET)*2) ;left most axis
DXAXISPAIR  SET ((SXAXISPAIR+DIGOFFSET)*2)   ;axis in center of display
DXAXISMAXPAIR  SET ((SXMAXPAIR+DIGOFFSET)*2) ;right most axis
DXSIZEPAIR  SET (DXMAXPAIR+1-DXMINPAIR)
DXSHALFPAIR SET (DXSIZEPAIR>>1)     ;one half the screen size

* Use these if the rectangle is in PAIR units, with border
DXMINPAIRBORD     SET (((DXMINBORDER*2)+3)&~3)  ;round up to nearest oct
DXMAXPAIRBORD     SET ((DXMAXBORDER*2)&~3)   ;round down to nearest oct
DXAXISMINPAIRBORD SET ((SXMINPAIRBORD+DIGOFFSET)*2)   ;left most axis
DXAXISPAIRBORD    SET ((SXAXISPAIRBORD+DIGOFFSET)*2)  ;axis in center of display
DXAXISMAXPAIRBORD SET ((SXMAXPAIRBORD+DIGOFFSET)*2)   ;right most axis
DXSIZEPAIRBORD    SET (DXMAXPAIRBORD+1-DXMINPAIRBORD)
DXSHALFPAIRBORD      SET (DXSIZEPAIRBORD>>1)    ;one half the screen size

*-----------------------
* Good amounts to move TDE rectangle on/off screen. 12*16 = 6*32 = 3*64 = 192
DXMOVE      SET 192
DXMHALF  SET (DXMOVE>>1)      ;one half the screen size
DXMOVEBORDER   SET DXMOVE     ;no diff
DXMHALFBORDER  SET (DXMOVEBORDER>>1)   ;one half the screen size
DXMOVEZIPPER   SET DXMOVE     ;no diff
DXMHALFZIPPER  SET (DXMOVEZIPPER>>1)   ;one half the screen size

DXMOVEPAIR  SET (DXMOVE*2)
DXMHALFPAIR SET (DXMOVEPAIR>>1)  ;one half the screen size
DXMOVEPAIRBORD SET DXMOVEPAIR    ;no diff
DXMHALFPAIRBORD   SET (DXMOVEPAIRBORD>>1) ;one half the screen size

*-----------------------------------------------------------------
*----- TDE VERTICAL SOURCE constants for non-trail TDE rectangle
SYMIN    SET (2+VERTSAFETY)   ;could useually be 1+
SYMAX    SET (238-VERTSAFETY) ;technically should be 239 !!!
SYAXIS      SET ((SYMAX+SYMIN)>>1)
SYSIZE      SET (SYMAX+1-SYMIN)

*----- TDE VERTICAL DESTINATION constants non-trail TDE rectangle
DYMIN    SET 1    ;avoid 1/2 scan line at top
DYMAX    SET 240     ;will only use 240 on field II
DYAXISMIN   SET SYMIN
DYAXIS      SET SYAXIS
DYAXISMAX   SET SYMAX
DYSIZE      SET (DYMAX+1-DYMIN)
DYSHALF     SET (DYSIZE>>1)

* Good amounts to move TDE rectangle on/off screen. 16*16 = 8*32 = 4*64 = 256
DYMOVE      SET 256
DYMHALF  SET (DYMOVE>>1)

***** DIGITAL TRAILS ON WRITE TDE ***************************************************************
```

```
* The source MIN/MAX actually define the limits where the video can be
* recorded, which is assumed to be the same as the coordinates as for a
* standard TDE source.
* The destination MIN/MAX, as always, are related to the Amiga screen.
* For Trails On Writing, you typically want to use the full screen.
*
*----- TDE HORIZONTAL SOURCE constants for digital trail TDE rectangle
SXMINDTW SET (5+HORSAFETY)
SXMAXDTW SET (189-HORSAFETY)
SXAXISDTW   SET ((SXMAXDTW+SXMINDTW)>>1)
SXSIZEDTW   SET (SXMAXDTW+1-SXMINDTW)

*----- TDE HORIZONTAL DESTINATION constants for digital trail TDE rectangle
DXMINDTW SET ZERO
DXMAXDTW SET (QUADSWIDE-1)
DXAXISMINDTW   SET (SXMINDTW+DIGOFFSETDTW)   ;left most axis
DXAXISDTW   SET (SXAXISDTW+DIGOFFSETDTW)  ;axis in center of display
DXAXISMAXDTW   SET (SXMAXDTW+DIGOFFSETDTW)   ;right most axis
DXSIZEDTW   SET (DXMAXDTW+1-DXMINDTW)
DXSHALFDTW  SET (DXSIZEDTW>>1)      ;one half the screen size

*----- TDE HORIZONTAL SOURCE constants for digital trail TDE rectangle, with zipper
SXMINDTWZIPPER SET (5+HORSAFETY)
SXMAXDTWZIPPER SET (189-HORSAFETY)
SXAXISDTWZIPPER   SET ((SXMAXDTW+SXMINDTW)>>1)
SXSIZEDTWZIPPER   SET (SXMAXDTW+1-SXMINDTW)

*----- TDE HORIZONTAL DESTINATION constants for digital trail TDE rectangle with zipper
DXMINDTWZIPPER    SET ZERO
DXMAXDTWZIPPER    SET (QUADSWIDE-1)
DXAXISMINDTWZIPPER   SET (SXMINDTWZIPPER+DIGOFFSETDTW)   ;left most axis
DXAXISDTWZIPPER      SET (SXAXISDTWZIPPER+DIGOFFSETDTW)  ;axis in center of display
DXAXISMAXDTWZIPPER   SET (SXMAXDTWZIPPER+DIGOFFSETDTW)   ;right most axis
DXSIZEDTWZIPPER      SET (DXMAXDTWZIPPER+1-DXMINDTW)
DXSHALFDTWZIPPER  SET (DXSIZEDTWZIPPER>>1)      ;one half the screen size

* Good amounts to move TDE rectangle on/off screen. 12*16 = 6*32 = 3*64 = 192
DXMOVEDTW   SET 192
DXMHALFDTW  SET (DXMOVEDTW>>1)      ;one half the screen size

*---------------------------------------------------------------
*----- TDE VERTICAL SOURCE constants for digital trail TDE rectangle
SYMINDTW SET (2+VERTSAFETY)   ;could usually be 1+
SYMAXDTW SET (238-VERTSAFETY) ;technically should be 239 !!!
SYAXISDTW   SET ((SYMAXDTW+SYMINDTW)>>1)
SYSIZEDTW   SET (SYMAXDTW+1-SYMINDTW)

*----- TDE VERTICAL DESTINATION constants non-trail TDE rectangle
DYMINDTW SET 1    ;avoid 1/2 scan line at top
DYMAXDTW SET 240     ;will only use 240 on field II
DYAXISMINDTW   SET SYMINDTW
DYAXISDTW   SET SYAXISDTW
DYAXISMAXDTW   SET SYMAXDTW
DYSIZEDTW   SET (DYMAXDTW+1-DYMINDTW)
DYSHALFDTW  SET (DYSIZEDTW>>1)

* Good amounts to move TDE rectangle on/off screen. 16*16 = 8*32 = 4*64 = 256
DYMOVEDTW   SET 256
DYMHALFDTW  SET (DYMOVEDTW>>1)

***** WIPES ************************************************************

*----- SOURCE constants for WIPE rectangle
* W1D_srcAxis
WSTART      SET ZERO
WEND     SET ($ffff)
WCENT    SET ($7fff)

*----- HORIZONTAL DESTINATION constants for WIPE rectangle
* W1D_position, W1D_destMin/Max/Axis/effectSize
WXMIN    SET ZERO
WXMAX    SET (PIXELSWIDE-1)
WXAXIS      SET ((WXMAX+WXMIN)>>1)
WXSIZE      SET (WXMAX+1-WXMIN)
WXSHALF     SET (WXSIZE>>1)
```

```
* Good amounts to move WIPE rectangle on/off screen. 48*16=24*32=12*64 = 768
WXMOVE      SET 768
WXMHALF  SET (WXMOVE>>1)

*----- VERTICAL DESTINATION constants for WIPE rectangle
WYMIN    SET ZERO
WYMAX    SET 480
WYAXIS      SET ((WYMAX+WYMIN)>>1)
WYSIZE      SET (WYMAX+1-WYMIN)
WYSHALF     SET (WYSIZE>>1)

* Good amounts to move WIPE rectangle on/off screen. 32*16=16*32=8*64 = 512
WYMOVE      SET 512
WYMHALF  SET (WYMOVE>>1)

* W1D_effectColor
KEYLOW      SET 0
KEYHI    SET 1

*****************************************************************
BGKEYQUAF   SET 256     ;use Black, White or QUAF
BGCOLORX SET 255     ;QUAF
BGCOLORY    SET 238  ;Line that contains a QUAF quad

BORDER      SET 252     ;border color Y, also use current border X
WHITE    SET 238     ;white quad ???

   ENDC  ;CROUTONS_I
```

CG 4.0 ARexx Documentation

Tentative ToasterCG ARexx Commands - October 18, 1994
by Arnie Cachelin

NOTE: These commands are subject to change, they may not all be fully implemented or bug free, and there will certainly be more forthcoming. If there are bugs, or missing commands please let me (Arnie) know ASAP.

GET_xxxx Commands

```
GET_CHAR()          -- Returns current character as a single letter string.
GET_CHAR(SPOT)      -- Returns current char position
GET_CHAR(FACE)      -- Returns current char's font name ***
GET_CHAR(SIZE)      -- Returns char's width (dx)
GET_CHAR(FIRST)     -- Returns first char on current line
GET_CHAR(LAST)      -- Returns last char on current line
GET_CHAR(NEXT)      -- Returns next char on current line
GET_CHAR(PREV)      -- Returns previous char on current line
GET_CHAR(RGBA)      -- Returns current char's (top) color
GET_CHAR(TOPR)      -- Returns current char's (top) color
GET_CHAR(BOTR)      -- Returns current char's bottom color

GET_BORD(SIZE)
GET_BORD(RGBA)
GET_BORD(PRIO)

GET_SHAD(TYPE)
GET_SHAD(PRIO)
GET_SHAD(SIZE)
// GET_SHAD(TOPR)
GET_SHAD(RGBA)
//GET_SHAD(BOTR)

GET_FONT()          -- Returns current char's font name
GET_FONT(TYPE)
GET_FONT(SIZE)
GET_FONT(name,[size]) -- Returns an available size of font named,

GET_RECT(SPOT)
GET_RECT(SIZE)
GET_RECT(PRIO)
GET_RECT(TOPR)
GET_RECT(RGBA)
GET_RECT(BOTR)

GET_PAGE()          // return number
GET_PAGE(FIRST)
GET_PAGE(LAST)
GET_PAGE(NEXT)
GET_PAGE(PREV)
GET_PAGE(TYPE)
GET_PAGE(SIZE)
GET_PAGE(BACK)
GET_LINE()
```

GET_LINE(FIRST)
GET_LINE(LAST)
GET_LINE(NEXT)
GET_LINE(PREV)
GET_LINE(TYPE)
GET_LINE(SIZE)
GET_LINE(JUST)
GET_LINE(SPOT)

SET_xxxx Commands

SET_ line, char, page, etc sets current object then PICK selects that object, or selects relative to that object (i.e. SET_ line number, PICK a char in that line). SET_ in the case of pages or attributes actually changes the page# or changes attrs of the selected object(s)

(change character values with Make/Kill char)
SET_CHAR(FACE,name)
SET_CHAR(TOPR,r,g,b,a)
SET_CHAR(BOTR,r,g,b,a)
SET_CHAR(RGBA,r,g,b,a)

SET_BORD(SIZE,#)
SET_BORD(RGBA,r,g,b,a)
SET_BORD(PRIO,T|B)

SET_SHAD(TYPE,D|C)
SET_SHAD(PRIO,T|B)
SET_SHAD(SIZE,#)
SET_SHAD(RGBA,r,g,b,a)

SET_RECT(SPOT,x,y)
SET_RECT(SIZE,x,y)
SET_RECT(PRIO,T|B)
SET_RECT(RGBA,r,g,b,a)

SET_PAGE(#)
SET_PAGE(FIRST)
SET_PAGE(LAST)
SET_PAGE(NEXT)
SET_PAGE(PREV)
SET_PAGE(TYPE)
SET_PAGE(BACK)

SET_LINE(#)
SET_LINE(FIRST)
SET_LINE(LAST)
SET_LINE(NEXT)
SET_LINE(PREV)
SET_LINE(TYPE)
SET_LINE(JUST,L|R|C|N)
SET_LINE(SPOT)

PICK_xxxx Commands

```
PICKPAGE([#])        // selects whole current page (jumps to it if # supplied)
PICKLINE([#])        // selects whole current line (sets new current if # supplied)
PICKCHAR([NOT])      //  Pick current char, deselect if arg provided
```

LOAD_xxxx Commands

```
<<LOADPAGE(name)>>
LOADBOOK(name)
LOADBRUSH(name)
LOADFONT(name)
LOADTEXT(name)
LOADPICT(name)
```

SAVE_xxxx Commands

```
<<SAVEPAGE(name)>>
SAVEBOOK(name)
SAVETEXT(name)
```

MAKE_xxxx Commands

```
MAKECHAR(letter)
MAKELINE(text)
MAKERECT(x,y,w,h)
MAKEPAGE([#])
MAKEBOOK(name)
```

KILL_xxxx Commands

```
KILLCHAR([#])
KILLLINE([#])
KILLPAGE([#])
{ KILLBOOK()         // maybe you should just delete a file!
```

Miscellaneous and Requester (REQ_xxxx) Commands

```
EXIT()               // Translates all pages into Farsi and pig-latinizes them before CG exits
RENDER([M|P])        // Main or Preview

REQ_FILE(title,[dir,[name]])
REQ_DIR(title,[dir])
REQ_STRING(title,[default])
REQ_NUMBER(title,[default])
REQ_ASK(title,[title2,[title3]])
REQ_TELL(title,[title2,[title3]])

REXX(macro)
MACRO(#,[macro])     // sets macro # to given string, or returns macro if none supplied
```

CG Book Format Documentation

I managed to dig this source out of the CG, it has a little comment about the book format, then 3 functions for reading lines in books created by different CG versions. The problem, in short, seems to be the fact that there may be an attribute structure appended (which is defined below) to each character's TextInfo structure.

```
/*
Book format on disk
-------------------
"ToasterBookFile4"
"FONT"
UWORD ID,StrLen,Height
"BRSH"
UWORD ID,StrLen
"DRAW"
UWORD ID,StrLen,Height,struct Draw
"PAGE"
UWORD Number
"LINE"
UWORD NumberText
UWORD Ascii
BYTE Kerning
UBYTE Attr?
struct Attributes
struct CGLine(from TotalHeight on)
"END!"
*/

#define BK_HEAD_SIZE          16
#define OLD_BOOK_VERSION      '1'
#define BOOK_VERSION          '3'
#define NEW_BOOK_VERSION      '4'
#define TYPE_SIZE              4

// make sure valid with book.h!
#define PAGE_SIZE             12

char BookHead[] = "ToasterBookFile";

BOOL __regargs ReadNewLine(
    struct CGLine       *Line,
    struct LockBuffer   *LB,
    UWORD               NumChars)
{
    ULONG               A;
    UBYTE               *Buff;
    struct              TextInfo *Text;

    Buff = (UBYTE *)RD->ByteStrip->Planes[0];

    Text = &Line->Text[0];
    if (!NumChars) {    // should never happen
        if (!(Text->Attr = AllocAttrib(&RD->DefaultAttr))) return(FALSE);
    } else
```

```
// read TextInfos
    for (A = 0; A < NumChars; A++) {
        if (4 != BufferedRead(Buff,4,LB)) return(FALSE);
        Text->Ascii = *(UWORD *)&Buff[0];
        Text->Kerning = Buff[2];
        if (Buff[3]) {
            if (!(Text->Attr = AllocAttrib(NULL))) return(FALSE);
            if (sizeof(struct Attributes) != BufferedRead(Text->Attr,
                sizeof(struct Attributes),LB)) return(FALSE);
        }
        Text++;
    }

// read CGLine
    if (LINE_ATTR_SIZE != BufferedRead(&Line->TotalHeight,LINE_ATTR_SIZE,LB))
        return(FALSE);

    return(TRUE);
}
//*****************************************************************************
BOOL __regargs ReadNotSoNewLine(
    struct CGLine       *Line,
    struct LockBuffer   *LB,
    UWORD               NumChars)
{
    ULONG               A;
    UBYTE               *Buff;
    struct TextInfo     *Text;
    struct Attributes3  *tmpattr;

    Buff = (UBYTE *)RD->ByteStrip->Planes[0];
    Text = &Line->Text[0];
    if (!NumChars) { // should never happen
        if (!(Text->Attr = AllocAttrib(&RD->DefaultAttr))) return(FALSE);
    } else
// read TextInfos
    for (A = 0; A < NumChars; A++) {
        if (4 != BufferedRead(Buff,4,LB)) return(FALSE);
        Text->Ascii = *(UWORD *)&Buff[0];
        Text->Kerning = Buff[2];
        if (Buff[3]) {
            if (!(tmpattr = (struct Attributes3 *)AllocAttrib(NULL))) return(FALSE);
            if (sizeof(struct Attributes3) != BufferedRead(tmpattr,sizeof(struct Attributes3),LB))
            {
                FreeAttrib((struct Attributes *)tmpattr);
                return(FALSE);
            }
            if (!(Text->Attr = AllocOldAttrib(tmpattr)))
            {
                FreeAttrib((struct Attributes *)tmpattr);
                return(FALSE);
            }
            FreeAttrib((struct Attributes *)tmpattr);
        }
        Text++;
    }
```

```
// read CGLine
    if (LINE_ATTR_SIZE != BufferedRead(&Line->TotalHeight,LINE_ATTR_SIZE,LB))
        return(FALSE);

    return(TRUE);
}

//*************************************************************************
BOOL __regargs ReadOldLine(
    struct CGLine       *Line,
    struct LockBuffer   *LB,
    UWORD               NumChars,
    struct CGPage       *Page)
{
    struct CGLine2      *OldLine;
    UWORD               A;
    struct TextInfo2    *TI2;
    UBYTE               *Buff;
    struct TextInfo     *Text;
    struct Attributes   *Attr;

    Buff = (UBYTE *)RD->ByteStrip->Planes[0];
    OldLine = (struct CGLine2 *)Buff;

// read TextInfos
    TI2 = (struct TextInfo2 *)Buff;
    Text = &Line->Text[0];
    for (A = 0; A < NumChars; A++) {
        if (sizeof(struct TextInfo2) != BufferedRead(Buff,
            sizeof(struct TextInfo2),LB))
            return(FALSE);
        Text->Ascii = TI2->Ascii;
        Text->Kerning = TI2->Kerning;
        Text++;
    }

// read CGLine
    if (LINE2_ATTR_SIZE != BufferedRead(&OldLine->FaceColor,LINE2_ATTR_SIZE,LB))
        return(FALSE);
    Line->TotalHeight = OldLine->TotalHeight;
    Line->Baseline = OldLine->Baseline;
    Line->XOffset = OldLine->XOffset;
    Line->YOffset = OldLine->YOffset;
    Line->Type = LINE_TEXT;
    Line->JustifyMode = OldLine->JustifyMode;
    if (A = OldLine->Seperator) {
        Line->Type = LINE_BOX;
        Line->FaceWidth = SepWidth[A-1];
        Line->FaceHeight = SepHeight[A-1];
        Line->Text[0].Ascii = 32; // one char faked for attr
    }
```

```
// read Text->[0].Attr
    if ((Page->Type != PAGE_CRAWL) || (NodesThisList(&Page->LineList) < 2)) {
        if (!(Attr = AllocAttrib(NULL))) return(FALSE);
        Line->Text[0].Attr = Attr;
        ConvertOldColor(&OldLine->FaceColor,&Attr->FaceColor);
        ConvertOldColor(&OldLine->ShadowColor,&Attr->ShadowColor);
        ConvertOldColor(&OldLine->OutlineColor,&Attr->OutlineColor);
        if (OldLine->RenderFont != 65535)
            Attr->ID = (OldLine->RenderFont >> 2);
        else
            Attr->ID = 0;
        if (OldLine->Seperator)
            Attr->ID = ID_BOX;
        Attr->ShadowLength = OldLine->ShadowLength;
        Attr->ShadowType = OldLine->ShadowType;
        if (OldLine->ShadowType > SHADOW_CAST) {
            Attr->ShadowType = OldLine->ShadowType - 2;
            Attr->ShadowColor.Alpha = OLD_TRANSP_ALPHA;
        }
        Attr->OutlineType = (OldLine->OutlineType >> 3); // convert to 0,1,..

    // somehow shadow direction meaning got switched
        Attr->ShadowDirection = OldLine->ShadowDirection + 4;
        if (Attr->ShadowDirection > 7) Attr->ShadowDirection -= 8;

        Attr->ShadowPriority = OldLine->ShadowPriority;
        Attr->SpecialFill=FILL_OLD;
        Attr->GradColor=Attr->FaceColor;
        Attr->OGradColor=Attr->OutlineColor;
    }
    return(TRUE);
}

////// cut out from book.h...
struct TextInfo {
    UWORD           Ascii;
    BYTE            Kerning;
    UBYTE           Select;
    struct Attributes   *Attr;
};

// Text[0]->Attr always has one of these to start line off
struct Attributes {
    struct TrueColor    FaceColor; // all colors treated as 1 attribute
    struct TrueColor    ShadowColor;
    struct TrueColor    OutlineColor;
    struct TrueColor    GradColor;
    struct TrueColor    OGradColor;
    UWORD           ID;
    UWORD           ShadowLength;
    UBYTE           ShadowType;
    UBYTE           OutlineType;
    UBYTE           ShadowDirection;
    UBYTE           ShadowPriority;
    ULONG           SpecialFill;
};
```

```
struct Attributes3 {
    struct TrueColor  FaceColor;
    struct TrueColor  ShadowColor;
    struct TrueColor  OutlineColor;
    UWORD             ID;
    UWORD             ShadowLength;
    UBYTE             ShadowType;
    UBYTE             OutlineType;
    UBYTE             ShadowDirection;
    UBYTE             ShadowPriority;
};

struct CGLine {
    struct MinNode    Node;
    struct TempInfo   Temp[LINE_LENGTH+1];
    struct TextInfo   Text[LINE_LENGTH+1];
    UWORD             TotalHeight;          // height of tallest char
    UWORD             Baseline;             // baseline of tallest font in line
    UWORD             XOffset;
    UWORD             YOffset;
    UWORD             FaceMinY;             // offset from top of render area to top of fonts
    UWORD             FaceWidth;
                      // for LINE_BOX,LINE_BRUSH,LINE_DRAW only
    UWORD             FaceHeight;
                      // for LINE_BOX,LINE_BRUSH,LINE_DRAW only
    UBYTE             Type;                 // IDs must match this Type for whole line
    UBYTE             JustifyMode;
    UBYTE             RenderPri;            // 0 is default & lowest, each Type different
    UBYTE             Rendered;             // only used during PageCall() TRUE/FALSE
};

#define LINE_ATTR_SIZE (sizeof(struct CGLine) - sizeof(struct MinNode) \
    - (sizeof(struct TempInfo)*(LINE_LENGTH+1)) \
    - (sizeof(struct TextInfo)*(LINE_LENGTH+1)))
```

NewTek Video Toaster CG - Front-End Processor Specification
Feb 9 1994
Prepared by Arnie Cachelin

The front-end processor (FEP) specification provided below will allow markets with languages requiring special keyboard input processing to use the Toaster CG through a customized FEP. The CG and the FEP communicate using standard Amiga message ports and ARexx messages. The set of commands required for FEP-CG interaction is limited although the FEP task may also send any other ARexx command supported by the CG. This minimum command set can be divided into 2 sets of commands, those sent by the CG to the FEP, and those sent by the FEP to the CG. Here they are:

Messages sent to FEP by CG

FEP_Begin
This message is sent when the user presses the hotkey in the CG if the FEP is available. The CG will close its interface window and supply the screen address in the message's argument (rm_Args[1]). The FEP should open its own interface window on the CG screen, return a "1" if this is successful, and start processing input.

FEP_Edit
This message is sent to the FEP if the hotkey is pressed and text is already selected. The command should be treated just like FEP_Begin, except that the current line is in the second argument (rm_Args[2]). The text is stored as a series of 2-byte codes separated by single null bytes. The length of this series is in the RexxArg structure. The screen address is still in rm_Args[1]. When the CG issues this command, the current line is left intact (with the cursor on the first letter). If the user aborts the FEP edit, the line will be unchanged, however if changes are made, the existing line will have to be altered or removed using other CG ARexx commands. Options for altering lines include KILLing the whole line, all the char.s one by one, moving to the altered character and setting its value, etc. If you have no reason to support the editing of existing text (e.g. Kanji), just ignore the extra data, and treat this command like FEP_Begin.

FEP_IntuiMsg
While the FEP is active any intuition messages sent to the CG are redirected to the FEP with the FEP_IntuiMsg message. This message contains any stray clicks or keypresses that the user makes on the CG screen itself. The address of a copy of the IntuiMessage structure received by the CG is in rm_Args[1]. The FEP should reply only to the ARexx message, not the IntuiMessage itself. One good response to a message like this is to make the FEP window active again, so keystrokes stop going to the CG window.

FEP_VerifyFont
This message is sent by the CG when loading a font. It allows the FEP to handle any market-specific font copy protection requirements. CG sends the name of the font file in rm_Args[1], and expects the FEP to return a 1 if the font is valid, or is unknown, or a 0 if the font is recognized but not legally installed. If CG gets a 0, the load will fail. A possible scenario for the implementation of this protection is as follows: User buys a font disk with an installer program. The installer program copies the font to the user's machine, and makes a change to the FEP itself to indicate that the newly installed font is valid. The FEP should already have a list of possible protected fonts, and the font installer will add a code to the relevant list entry showing that that font can be legally loaded.

Messages sent by FEP to CG

FEP_End
This message tells the CG that the FEP has closed its interface and the CG should resume normal operation. The CG will not end any messages to the FEP after this command is received until the next hotkey press.

FEP_NewChar

This message is sent by the FEP when the user has confirmed a character choice. The argument should contain the appropriate 2-byte code.

FEP_LoadFont

This message can be sent to the CG to load a composite postscript font. If a filename and size are provided this font will be loaded at the given size, otherwise the file requester pops up. The supplied font name should be the complete path and name of one of the type 1 fonts (i.e. ToasterFonts/Mincho/MinchoA3). The CG will then load ALL the fonts from that font set. By default, the composite fonts are assumed to have 2 Hex digits appended to the main font name. The Hangul type 1 fonts apparently use 3 Decimal digits (i.e. Kim/Kim011). This can be handled using the Type parameter. The bits of this number determine whether the font is a composite font (bit 0), and whether it uses 2 hex or 3 dec digits (bit 1):

```
PS_Normal    = 0
PS_Composite = 1
PS_Hex       = 2
PS_Dec       = 0
FEP_LoadFont("Mincho/MinchoA3",40,PS_Composite+PS_Hex)
FEP_LoadFont("Kim/Kim001",40,PS_Composite+PS_Dec)
```

General Interface Notes

The CG's ARexx port name is 'CG_AREXX' and the FEP's port is named 'FEP.PORT'. These names are case sensitive. Since both CG and FEP use ARexx standard messages, it is simple to create ARexx scripts which simulate the functions of these two programs. Such scripts are invaluable debugging tools and they will be provided along with the FEP example code. The FEP should already be running when the CG is launched.

ColorFonts for the Video Toaster Character Generator

revised and updated by Daniel Wolf
16 October 1995

ColorFonts are anti-aliased sixteen million color fonts. ColorFonts are capable of having photorealistic imagery, using all sixteen millions colors at once, but they are typically quite large, and (currently) the user is unable to modify them.

Creating ColorFonts

ColorFonts are sixteen million color ToasterFonts (not to be confused with Amiga ColorFonts). ColorFonts are currently created by:

1. Saving an IFF brush for each character in the font
2. Writing a text file which contains the filenames of each brush
3. Processing the brushes with a CLI utility called "colortoast"

Colortoast reads the script all the brush file names and creates an anti-aliased ColorFont from the individual brushes. Colortoast is not included with the Toaster because it may be difficult for many Toaster users. It is available to Video Toaster developers only, directly from NewTek.

Here's the procedure for creating your own Toaster ColorFonts, expanded from the brief list described above.

1. Create a set of 89 IFF 24-bit images of up to 160x160 pixels with the names Pic033, Pic034, ...Pic120, Pic121 and save them all in one directory, CFPics: (an 'assigned' directory). An ideal tool for this step is LightWave3D (more about that later). If you create the images as ordinary Amiga graphics (2-256 colors), you'll want to process them into 24-bit images with Art Department Professional or an equivalent tool.

2a. A pre-written text file of all your images, named CFInList, is part of this ColorFont-making package and is installed in your s: directory. It looks like this:

```
CFPics:Pic033
CFPics:Pic034
.
.
.
CFPics:Pic120
CFPics:Pic121
```

This CFInList file list will come into play in Step 3.

2b. Another pre-written text file, CFOutList, is also supplied with this package. It comes into play in the Step 3 also. CFOutList looks like this:

```
ram:CF/CFBrush.33
ram:CF/CFBrush.34
.
.
.
ram:CF/CFBrush.120
ram:CF/CFBrush.121
```

In CFOutList, the .33 means ASCII 33 (an exclamation point), .65 means ASCII 65 (an uppercase A), etc. When the set of brushes created in the next step is processed into a font, the images you start with will become the characters from ASCII 33 up to ASCII 121. If you start with images of characters, you'll get a ColorFont of characters. If you start with other kinds of images, then when you use that ColorFont in the Toaster CG, you'll get those images instead of characters!

3. Execute the MakeCFont80 script also supplied with this set of tools (and also installed into your s: directory). MakeCFont is a two-line AmigaDos script which first calls an ARexx script caled FontCutter.rexx (again, part of this package and also installed in your s: directory).

FontCutter.rexx uses a program called BrushCutter (included in this package also, and installed into your c: directory) to cut out 24-bit brushes from your set of images. FontCutter.rexx reads in each image name listed in CFInList, tells BrushCutter to process that image into a brush, and then saves the brush according to the list of file names in CFOutList. BrushCutter intelligently cuts out brushes of imagery from a black background. When the images are pictures of characters in a font character set, BrushCutter's intelligently cuts the brushes to appropriate sizes for use in a ColorFont.

All of the brushes created by FontCutter.rexx are placed into your ram: disk, in a directory called CF.

While it applies BrushCutter to each of your original 24-bit images, FontCutter.rexx also builds a third temporary script, ram:CFScript. When FontCutter.rexx is finished turning your images into brushes, MakeCFont80 calls ColorToast to process the set of brushes into a finished Toaster ColorFont in your ram: disk, called CFont.80.

When MakeCFont80 is finished performing its two steps (FontCutter.rexx and ColorToast), you are ready to copy your CFont.80 80-point ColorFont to a hard drive or floppy disk and use it with the Toaster CG.

Voila!

If you wish to build the usual complement of 40-point, 60-point and 80-point versions of a single ColorFont, you'll want to:

1. Use MakeFont80 and 160x160 images for a 80-point ColorFont.
2. Use MakeFont60 and 120x120 images for a 60-point ColorFont.
3. Use MakeFont40 and 80x 80 images for a 40-point ColorFont.

Using LightWave to Create the Original Image Set

You've also received two more tools for ColorFont creation via LightWave:

Color_Font.lwm	a LightWave Modeler ARexx Macro
Color_Font.lws	a LightWave Scene file

If you installed this package from the AFX_CF disk, the Color_Font.lwm macro is in your s: directory (it is an ARexx script for Modeler) and Color_Font.lws is in your Work:CFontLWTools directory (along with this Doc file).

Here's a step-by-step description of one way (there are definitely more variations you can work out for your own convenience) to use LightWave and these ColorFont tools.

A. Building a LightWave Object of a Character Set with the Color_Font.lwm Modeler Macro.

1. Start LightWave.
2. Enter Modeler.
3. Select the Text Gadget (lower left part of Modeler screen).
4. When the Generate Text panel opens, note that the Font popup says ' (none) '.
5. Select the Load Gadget next to the Font popup.
6. Use the requesters to select a PostScript-type font to work with.
7. When you've brought in a font, make sure you use the Font popup to specify it and that the (none) designation is gone.
8. Exit the Generate Text panel by clicking on its OK button.
9. Go to the Amiga Workbench (Left-Amiga-M).
10. Start an Amiga Shell.
11. Type in: rx Color_Font.lwm .
12. The Macro will bring up a panel and you can observe that the font you specified is named in the panel. The panel also shows that you'll build an LightWave object using characters 33 to 121 of the font. You can click on the Bevel button gadget if you want the LightWave object to have beveled edges.
13. Click on the OK button on the Macro's panel.
14. Now wait and observe the Color_Font.lwm Modeler Macro build a series of characters.
15. When the Macro is complete (in a few minutes, perhaps) you can observe its Done button and click on it.
16. Now Click on Modeler's Save As Gadget.
17. Use the file requesters to save the Light Wave Object which was just created by the Color_Font.lwm Macro. For example, if you used CalgaryDemiBold as your font, you might want to save the object you made as CalDemi in your Objects directory.

You've just created a large LightWave object with all the 89 characters of a ColorFont character set - all laid out in a long line and spaced appropriately (on a 2 meter Grid) for use in the next step.

B. Rendering a Set of 89 Character Images to Process with MakeFont80 ColorFont-creating Script.

1. Go to LightWave's Layout screen.
2. Click on the Scene Menu gadget at the top left of the screen.
3. When the Scene panel appears, click on the Load button gadget.
4. Load the scene named Color_Font.lws from your Work:CFontLWTools directory.
5. LightWave will alert you that it cannot find the Font.lwo object and offer you an opportunity to load an alternative object. Click on the OK button and load in the object you just created in Step A. above (e.g. CalDemi.lwo).
6. When the scene is loaded in, exit the Scene panel.
7. If you specify a Camera view and step through the frames of the LightWave scene you just loaded, you'll see that each frame will render an image of one of the characters of the character set used to build the LightWave object when you invoked Color_Font.lwm!
8. Click on the Images button at the top of the Layout screen and you'll enter the Images Panel. Load in an image of your choice then exit the Images Panel.
9. Now comes the artistic part. Enter the Surfaces panel and you'll find that all your object has the Default surface. Click on the T next to the Surface Color option.
10. When the Surface Texture panel appears, specify a Planar Texture Map on the Z axis and set the Texture Size to X=2, Y=2, and Z=1. Use the Image popup to specify the image you just loaded in the Images Panel, then exit the Surfaces Panel.
11. Click on the Record button at the top of the Layout screen.
12. Select the option to Save RGB Images in the Record Panel. When the File Requester appears, specify that you want to save to your CFPics: (assigned) directory, with a picture prefix of Pic. Then exit the Record Panel.

13. Click on the Camera button at the top of the Layout screen.
14. Specify a Custom Size of 160x160 (even up to 240x160 will be OK here) and then exit the Camera Panel.
15. Now Render the scene. LightWave will create a series of images of your Character Set with your selected image 'texture-wrapped' onto each letter.
16. When LightWave is finished with the rendering, you can use the set of 160x160 images you just created to process into a ColorFont with the MakeCFont80 script described earlier.

There you have it. You can use the CFontLWTools to automate a large part of the work of building your 160x160 images for conversion to a 80-point Toaster ColorFont with the tools described at the beginning of this doc file.

Have fun!

For more information on developing software and hardware products for the Video Toaster, please contact NewTek.

NewTek, Inc.
Developer Support Program
1200 S.W. Executive Drive
Topeka, KS 66615
913-228-8000

For your reference:

Partial ASCII Character Set

SP	32	@	64	‘	96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
’	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105
*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
-	45	M	77	m	109
.	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92	|	124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95		

```
;Flyer-Toaster ES Message Programming Example
;by Daniel Wolf 7/4/95
;for MACRO68 Assembler
;(c) Copyright 1995 by NewTek, Inc.

 EXEOBJ
 RELAX
 MC68000
 NOCOMMENTMARKERS

 JMP _START                              ;jump past includes, etc.

;**************** Preliminaries

;*** Intuition Equates - needed for includes

CUSTOMSCREEN        EQU $F
ACTIVATE            EQU $1000
DISKINSERTED        EQU $8000
DISKREMOVED         EQU $10000
MOUSEBUTTONS        EQU $8
INTUITICKS          EQU $400000
NOCAREREFRESH       EQU $20000
RAWKEY              EQU $400
GADGETDOWN          EQU $20
GADGETUP            EQU $40
RMBTRAP             EQU $10000
SMART_REFRESH       EQU $0
BORDERLESS          EQU $800
BACKDROP            EQU $100
BOOLGADGET          EQU $1
GADGHNONE           EQU $3
GADGIMMEDIATE       EQU $2

;*** Amiga Includes

 INCLUDE "INCLUDES:EXEC.I"              ;types.i, ports.i, nodes.i, etc. in one
 INCLUDE "INCLUDES:GADGET.I"            ;Amiga Gadget basis for FastGadgets

;*** Toaster Includes

 INCLUDE "INCLUDES:ELH.I"               ;needed for INSTINCT.I
 INCLUDE "INCLUDES:VTHAND.I"            ;needed for INSTINCT.I
 INCLUDE "INCLUDES:EDITSWIT.I"          ;Editor-Switcher Message Definitions
 INCLUDE "INCLUDES:INSTINCT.I"          ;reference to TB_ARexxPort

;*** Private Structure for Local Variables

SOFFSET SET 0
 LONG _STACK
 LONG _DOSBASE
 LONG _TOASTBASE
 LONG COMMAND
 LONG STDOUT
 LONG PUBPORT
 LONG FG                                ;the FastGadget of the Crouton
SOFFSET SET 0

;*** Amiga Exec and DOS Lib Offsets

LVO.CLOSELIBRARY    EQU $FFFFFE62
LVO.FINDTASK        EQU $FFFFFEDA
LVO.OPENLIBRARY     EQU $FFFFFDD8
LVO.WAITPORT        EQU $FFFFFE80
LVO.REMOVE          EQU $FFFFFF04
LVO.ADDPORT         EQU $FFFFFE9E
LVO.PUTMSG          EQU $FFFFFE92
LVO.FREESIGNAL      EQU $FFFFFEB0
```

```
LVO.ALLOCSIGNAL         EQU $FFFFFEB6

LVO.OUTPUT              EQU $FFFFFFC4
LVO.WRITE               EQU $FFFFFFD0

;*** Some Useful Macros

SYSLIB MACRO
 move.l $4,a6
 jsr LVO.\1(a6)
 ENDM

DOSLIB MACRO
 move.l _DOSBASE(a5),a6
 jsr LVO.\1(a6)
 ENDM

JUST MACRO
 jsr LVO.\1(a6)
 ENDM

DOSPRINT MACRO                          ;*filehandle (d1),*buff (d2), [len (d3)]
 movem.l d0-d3/a0-a6,-(SP)
 move.l \1,d1
 move.l \2,d2
 move.l d2,a0                           ;put length of null-terminated
CALC\@                                  ;string into d3 for DOS WRITE command
 tst.b (a0)+
 bne.s CALC\@
 move.l a0,d3
 sub.l d2,d3
 subq.l #1,d3
 DOSLIB WRITE
 movem.l (SP)+,d0-d3/a0-a6
 ENDM

;**************** Program Code

_START                                  ;Amiga Startup Stuff
 lea VARIABLES,a5
 move.l SP,_STACK(a5)                   ;save stack pointer
 move.l $4,a6
 move.l a0,COMMAND(a5)                  ;save ptr to the clip name!
 cmpi.l #4,d0                           ;command at least 4 characters long!
 bmi _STARTERROR
 clr.b -1(a0,d0.W)                      ;null-terminate the command line

 lea _DOSNAME,a1
 moveq #34,d0
 JUST OPENLIBRARY
 move.l d0,_DOSBASE(a5)                 ;check for successful open
 beq _STARTERROR

 lea _TOASTNAME,a1
 moveq #0,d0
 JUST OPENLIBRARY
 move.l d0,_TOASTBASE(a5)
 beq _STARTERROR

 DOSLIB OUTPUT
 move.l d0,STDOUT(a5)                   ;set CLI window as OUTPUT file handle

NOWDOMAIN
 DOSPRINT STDOUT(a5),#TITLE             ;print out program title
 DOSPRINT STDOUT(a5),COMMAND(a5)        ;print out clip name from command line
 DOSPRINT STDOUT(a5),#TWOLINE           ;do 2 linefeeds and carriage returns
 jsr CROUTONIT                          ;do the Flyer routines
```

```
_ERROR
 move.l d0,-(SP)                    ;clean up and exit to AmigaDos
 move.l $4,a6
 move.l _TOASTBASE(a5),d0
 beq.s 5$
 move.l d0,a1
 JUST CLOSELIBRARY
5$
 move.l _DOSBASE(a5),d0
 beq.s 9$
 move.l d0,a1
 JUST CLOSELIBRARY
9$
 move.l (SP)+,d0
 move.l _STACK(a5),SP               ;restore stack pointer
 rts                                ;exit back to where we came from!

_STARTERROR
 moveq #$ffffffff,d0
 bra _ERROR

CROUTONIT
 bsr SETUPPORT
 tst.l d0
 beq ERROR                          ;didnt get our ESMSG port!

 lea MESSAGE,a1                     ;load the crouton
 move.w #ES_LoadCrouton,ES_Type(a1)
 move.l COMMAND(a5),ES_Data1(a1)
 move.l #1,ES_Data2(a1)
 bsr SENDIT
 move.l ES_Reply(a1),FG(a5)
 beq.s DONE

 move.w #ES_Select,ES_Type(a1)      ;select this crouton
 move.l FG(a5),ES_Data1(a1)
 bsr SENDIT

 move.w #ES_Auto,ES_Type(a1)        ;auto/execute the crouton
 move.l FG(a5),ES_Data1(a1)
 bsr SENDIT

 move.w #ES_FreeCrouton,ES_Type(a1) ;and now free the crouton
 move.w FG(a5),ES_Data1(a1)
 bsr SENDIT

DONE
 tst.l PUBPORT(a5)                  ;did we make a port?
 beq.s 1$
 move.l PUBPORT(a5),a1
 SYSLIB REMOVE                      ; if so - remove it from list
 move.l PUBPORT(a5),a0
 moveq #0,d0
 move.b MP_SIGBIT(a0),d0            ; and free the ports SIGBIT
 JUST FREESIGNAL
1$
 moveq #0,d0                        ;all is well
ERROR
 rts
```

```
SENDIT
 move.w #EditCookie,ES_Cookie(a1)          ;abracadabra
 move.l PUBPORT(a5),MN_REPLYPORT(a1)       ;reply to me!
 move.b #NT_MESSAGE,LN_TYPE(a1)
 move.w #128,MN_LENGTH(a1)
 move.l _TOASTBASE(a5),a2
 move.l TB_ARexxPort(a2),a0                ;port in a0
 SYSLIB PUTMSG
0$
 move.l PUBPORT(a5),a0
 JUST WAITPORT
 cmpi.l #MESSAGE,d0                        ;is it our message coming back?
 bne.s 0$                                  ;stupid wait loop
 move.l d0,a1
 cmpi.b #NT_REPLYMSG,LN_TYPE(a1)           ;is this a reply to our message?
 bne.s 0$
 rts

SETUPPORT
 move #-1,d0                               ;don't care which sigbit this port gets
 SYSLIB ALLOCSIGNAL                        ;get a sigbit
 move.b d0,d7                              ;stash sigbit
 cmpi.b #-1,d0                             ;if error -
 beq ERR_PORT1                             ; branch out from here

 lea PORTMEM,a4
 suba.l a1,a1
 JUST FINDTASK                             ;find the task control block and
 move.l d0,MP_SIGTASK(a4)                  ;tell port where the tcb is
 move.l #PORTNAME,LN_NAME(a4)              ; and its name
 move.b #0,LN_PRI(a4)                      ; and its priority
 move.b #NT_MSGPORT,LN_TYPE(a4)            ; and its type
 move.b #PA_SIGNAL,MP_FLAGS(a4)            ; and its flags
 move.b d7,MP_SIGBIT(a4)                   ; and which sigbit it got
PUBLICPORT
 move.l a4,a1                              ;anything but PUBLIC would be DUMB
 JUST ADDPORT                              ;tell ADDPORT where the port is
 move.l a4,PUBPORT(a5)                     ;and ADD it
 move.l a4,d0                              ;and tell this program where it is
 rts                                       ;and return its address
ERR_PORT1
 moveq #0,d0                               ;if error, set d0 = 0
 rts                                       ;signal allocation failed

;**************** Data Section

 DATA

 CNOP 0,4
_TOASTNAME dc.b 'ToasterBase',0
 EVEN

_DOSNAME dc.b 'dos.library',0
 EVEN

PORTNAME dc.b 'ESTEST',0
 EVEN

TITLE   DC.B 13,10,' ESTest by D. Wolf    1995 by NewTek'
TWOLINE  DC.B 10,13,10,13,0,0,0,0
 CNOP 0,4

VARIABLES     dx.l 20,0                   ;private (STACK, COMMAND, CMDLEN, etc.)
MESSAGE       dx.l 64,0                   ;mem for our message
PORTMEM       dx.l 20,0                   ;mem for our message port

END
```

```
*****************************************************************
* $EditSwit.i$
*****************************************************************
    IFND EDIT_SWIT_I
EDIT_SWIT_I SET 1

    IFND EXEC_TYPES_I
    INCLUDE "exec/types.i"
    ENDC
    INCLUDE "exec/ports.i"

*   FG_Extra1
CR_Overlap   equ  0
CR_VidEvent equ  1
CR_Other     equ  2

EditCookie    equ  $666

MAX_ESPARAMS        equ     8

*-------------------------------------------------------
    STRUCTURE ESMessage,MN_SIZE
    WORD   ES_Cookie          ;$666
    WORD   ES_Type
    LONG   ES_Reply       ;Usually returns a useful number, pointer, or error number
    APTR   ES_Error       ;->some type off Error structure

    LONG   ES_Data1
    LONG   ES_Data2
    LONG   ES_Data3
    LONG   ES_Data4
    LONG   ES_Data5
    LONG   ES_Data6
    LONG   ES_Data7
    LONG   ES_Data8       ;actually any number of LONGs is allowed
    LABEL  ES_SIZEOF      ;so, look at MN_LENGTH for total length

*-------------------------------------------------------
    STRUCTURE RenderCallBack,0
    APTR   RCB_Function
    APTR   RCB_FG
    ULONG  RCB_Frame
    ULONG  RCB_Min
    ULONG  RCB_Max
    ULONG  RCB_Flags      ;bit 31 = initial FF/REW direction (1=reverse)
                          ;bit 30 = clear if FF/REW, set if Shuttle
    APTR   RCB_Window
    APTR   RCB_Gadget
    WORD   RCB_MouseY ;initially same as sc_MouseY
    WORD   RCB_MouseX ;initially same as sc_MouseX

* Velocity = RCB_VelocityNumerator/RCB_VelocityDenominator
    WORD   RCB_VelocityNumerator
    UWORD RCB_VelocityDenominator

    LABEL   RCB_SIZEOF
```

```
    BITDEF  DHD,DHD_PLAY_REV,31          ; initial shuttle direction (1=reverse)
    BITDEF  DHD,DHD_SHUTTLE_MODE,30      ; clear if FF/REW Jog, set if Shuttle
    BITDEF  DHD,INPOINT,29               ; set if moving inpoint
    BITDEF  DHD,OUTPOINT,28              ; set if moving outpoint
    BITDEF  DHD,VIDEOSLIDER,27           ; set if moving video slider
    BITDEF  DHD,AUDIOSLIDER,26           ; set if moving audio slider

*------------------------------------------------
HACK_TWEAK      EQU 'TWEK'              ;TWEK  first hack type
HACK_TBCwrite   EQU 'TBCW'
HACK_TBCread    EQU 'TBCR'
HACK_TBCopen    EQU 'TBCO'
HACK_TBCclose   EQU 'TBCC'

    STRUCTURE TweakHack,0
     ULONG      twhk_Flags
     ULONG      twhk_Position            ; -910 -> 910
     ULONG      twhk_Clock               ; 0 -> 3
     ULONG      twhk_Coarse              ; 0 -> 15
     ULONG      twhk_Fine                ; 0 -> 7
     ULONG      twhk_Luma                ;
     ULONG      twhk_Hue                 ;
     ULONG      twhk_Pedestal            ; 0 -> 255
     ULONG      twhk_Shift               ; 0 -> 1
  LABELtwhk_SIZEOF

;twhk_Flags bits
HKB_PLAY_A      EQU 0
HKB_PLAY_B      EQU 1
HKB_RECORD_A    EQU 2
HKB_RECORD_B    EQU 3
HKB_TESTREC     EQU 4
HKB_READCALIB   EQU 5

HKF_PLAY_A      EQU 1
HKF_PLAY_B      EQU 2
HKF_RECORD_A    EQU 4
HKF_RECORD_B    EQU 8
HKF_TESTREC     EQU 16
HKF_READCALIB   EQU 32

*------------------------------------------------
    STRUCTURE SystemPrefs,0
     UBYTE      spref_Termination
     UBYTE      spref_GPI                ;0=off, 1=positive, 2=negative
     UBYTE      spref_Flags1
     UBYTE      spref_Flags2
    LABEL       spref_SIZEOF

* Flags1 bits
    BITDEF  sp,PrvwOLay,0       ;set if Preview Overlay is on, else 3 monitor setup
    BITDEF  sp,FlyerVID3,1      ;set if Toaster VID3 is connected to Flyer, else it's External
    BITDEF  sp,FlyerVID4,2      ;set if Toaster VID4 is connected to Flyer, else it's External
    BITDEF  sp,FastDrive,3      ;set if extra fast drives are present, fast recording enabled
```

```
* Termination Setting:
      BITDEF  sp,Terminate4,0          ;set if Video 4 input is terminated
      BITDEF  sp,Terminate3,1          ;set if Video 3 input is terminated
      BITDEF  sp,Terminate2,2          ;set if Video 2 input is terminated
      BITDEF  sp,Terminate1,3          ;set if Video 1 input is terminated

*------------------------------------------------------
* This will record the currently selected VideoSource & AudioSources, using
* the current compression mode.  See ES_CompressionMode, ES_RecordSource

*+ NOTE!!! This struct is now part of the ES command structure (starting at Data1)
*+     STRUCTURE FlyerRecord,0
*+   APTR     FLY_Name            ;->path/name
*+   ULONG  FLY_Fields ;length of record, 0=wait for abort
*+
*+* The following functions are ignored if NULL.
*+* They are used to start and stop record frame accurately.
*+* So record start and stop could be controled by GPI, input SMPTE time,
*+* Time of Day, Keyboard or Mouse input, etc.
*+* Start and stop will take affect on start of the next color frame (field 1).
*+   ULONG  FLY_StartFunction ;Will not start until StartFunct returns
*+   ULONG  FLY_StopFunction  ;Will not stop until Stop function returns
*+   ULONG  FLY_Flags    ;see below
*+     LABELFLY_SIZEOF

*** FLY_Flags ***
      BITDEF  FLY,APPEND,0             ;For appending to existing clips
      BITDEF  FLY,SERIAL1,1            ;Include Serial port1 data
      BITDEF  FLY,SERIAL2,2            ;Include Serial port2 data

*------------------------------------------------------
* Values used by ES_RecordSource(VideoSource,,,)
FLY_VideoSource_NONE        EQU  0      ;Show black, don't record video
FLY_VideoSource_NTSC        EQU  1      ;Requires Flyer TBC
FLY_VideoSource_SVHS        EQU  2      ;Requires Flyer TBC
FLY_VideoSource_VID1        EQU  3
FLY_VideoSource_VID2        EQU  4
FLY_VideoSource_Main        EQU  5
FLY_VideoSource_Preview     EQU  6
FLY_VideoSource_VID3        EQU  7
FLY_VideoSource_VID4        EQU  8

*------------------------------------------------------
* Values used by ES_CompressionMode
* I don't know what modes we'll support, so I just made up some names!!!
FLY_CompressionMode_D2              EQU 0
FLY_CompressionMode_BetaCam         EQU 1
FLY_CompressionMode_SVHS            EQU 2
FLY_CompressionMode_VHS             EQU 3
FLY_CompressionMode_BroadCast       EQU 4
FLY_CompressionMode_Industrial      EQU 5
FLY_CompressionMode_ThreeQuarter    EQU 6
FLY_CompressionMode_High8           EQU 7
FLY_CompressionMode_PreviewQuality  EQU 8
```

```
*-----------------------------------
* Flags used by ES_LocateFile & ES_FoundFile
StringGadget  equ  1              ;set if you want a string gadget (for save)


*****************************************************************
* See edit/inc/editswit.h for more comments.  Or switcher/src/pecomm.a

* ESMessage.Type
ES_DUMMY            equ   0
ES_STARTUP          equ   1       ;Data1->TB_Screen, Data2=NULL,
                                  ;Data3->TB_Window, ;Data4->table of pointers to FG
                                  ;lists
ES_RENDER_EDIT      equ   2       ; tells Editor to (re)render its windows
                                  ; Data1: TRUE means open windows if not open (and
                                  ;refresh)
                                  ; FALSE means close windows (no refresh before
                                  ;close)
                                  ; (WINDOWOPENFLAG, ScreenDepth)  ;ONLY
                                  ;SENT TO EDITOR, NEVER TO SWITCHER!
ES_RENDER_SWIT      equ   3       ; tells Switcher to (re)render, Data1: same
                                  ;ES_RENDER_EDIT
ES_ClearProject     equ   4       ; Clear a given project list
ES_PanelOpen        equ   5       ; Called when control panel is opened. USED TO BE
                                  ;SETRMOFF
ES_GETPLIST         equ   6       ; asks switcher for project list pointers, reply will be
                                  ;pointer to table of pointers to FG lists

    ; This info is also supplied by the ES_STARTUP message!!!

ES_FreeCrouton      equ   7       ; asks switcher to unload crouton from memory
ES_LoadCrouton      equ   8       ; asks switcher to load crouton data1 is pointer
                                  ; to disk file name.
ES_DuplicateCrouton equ   9       ; make a duplicate of the crouton. Data1 is a pointer
                                  ; to the crouton to duplicate
ES_LoadProject      equ   10      ; Data1 is filename Data2 is mode
ES_SaveProject      equ   11      ; Data1 is filename or Null for current file
ES_NewProject       equ   12      ; Data1 is filename of new file
ES_Select           equ   13      ; Data1 is FG to get FGC_SELECTQ
ES_Auto             equ   14      ; Data1 is FG to get FGC_AUTO
ES_StartSeq         equ   15      ;
ES_Stop             equ   16      ; used to stop playing/recording clips & sequencing
ES_QUIT             equ   17      ; Tells the Editor to quit = F8
ES_GetValue         equ   18      ; Gets value out of taglist
ES_PutValue         equ   19      ; Puts value into taglist
ES_FGcommand        equ   20      ; Sends a specific FGC command to crouton
ES_PanelClose       equ   21      ; Called when control panel is closed. Was
                                  ;ES_HiliteCrouton
ES_Record           equ   22      ; (->name, fields, ->StartFunct, ->StopFunct, Flags)
ES_Pause            equ   23      ; (PauseFlag)
ES_Jog              equ   24      ; (RenderCallBack)
ES_Shuttle          equ   25      ; (RenderCallBack)
ES_InitRecord       equ   26
ES_InitPlay         equ   27
ES_Main2Blank       equ   28
```

```
ES_GUImode          equ   29    ; Project/Switcher, Project/Files etc.
ES_SelectDefault    equ   30    ; Select internal default effect (fade)
ES_Jump             equ   31    ; (FG,field)
ES_ApplyTags        equ   32    ; (FGsource,FGdestination)
ES_TagSize          equ   33    ; (FG, TagID)
ES_GetTable         equ   34    ; (FG, ->Table, TableSize, TagID)
ES_PutTable         equ   35    ; (FG, ->Table, TableSize, TagID)
ES_LightWave        equ   36    ; (FGC_Command)
ES_ToasterPaint     equ   37    ; (FGC_Command)
ES_ToasterCG        equ   38    ; (FGC_Command)
ES_ChromaFX         equ   39    ; (FGC_Command)
ES_FlyerDriveInfo   equ   40    ; (->volumename, ->FlyerVolInfo or NULL)
ES_RecordSource     equ   41    ; (VideoSource)
ES_CompressionMode  equ   42    ; (mode)
ES_CheckRecord      equ   43    ; (), returns FERR_..., 0 if recording finished OK
ES_DefragFlyer      equ   44    ; (->volumename)
ES_StartHeadList    equ   45    ; ()
ES_MakeClipHead     equ   46    ; (->name,VidStart,VidFields,AudStart,AudFields)
ES_EndHeadList      equ   47    ; ()
ES_MakeClipIcon     equ   48    ; (->name,->crud,crudsize,field)
ES_FlyerClipInfo    equ   49    ; (->clipname, ->ClipInfo or NULL)
ES_AppendIcon       equ   50    ; (->name)
ES_Hack             equ   51    ; ('Name',->HackStructure)
ES_BuildVolumeTable equ   52    ; () returns ->table of Flyer Volumes & Type
ES_GetPrefs         equ   53    ; (->SystemPrefs Structure) fill in current values
ES_SetPrefs         equ   54    ; (->SystemPrefs Structure)
ES_LoadedSlices     equ   55    ; () returns Slice Mask (see tags.i for bits)
ES_UnSavable        equ   56    ;(FG, TagID);marks Tag item so it doesn't get saved in the
                                ;project
ES_NewFieldCount    equ   57    ;(FG);This needs to be called if a Current FXs speed or
                                ;duration has changed.
ES_SavePrefs        equ   58    ;() saves an HS file. Returns ERROR. Though we usually
                                ;ignore errors!
ES_StartClipCutList equ   59    ;(->filename, DestructFlag)
ES_AddClipCut       equ   60    ;(->filename, StartField, NumFields, VidAudFlags)
ES_EndClipCutList   equ   61    ;(DoItFlag)
ES_AudioControl     equ   62    ;(->FlyAudCtrl, operationFlags)
ES_ChangeAudio      equ   63    ;(->FG)  Use this to modify the currently playing clip
ES_LocateFile       equ   64    ;(Type,->Name,flags,->textarray,mode)
ES_FoundFile        equ   65    ;(Type,->Name,flags,->textarray,mode)
ES_SwitcherRAWKEY   equ   66    ;(Code,Qualifier)
ES_RecordAppend     equ   67    ; (->name, fields )

*-----------------------------------------

* ESMessage.Reply
ES_ERR_NONE         equ   0
ES_ERR_GENERIC      equ   1     ; unable to complete requested action

    ENDC ; EDIT_SWIT_I
```

Toaster/Switcher ARexx Documentation

Video Toaster Switcher Arexx Manual 4/20/95

****

Update for 4.04 on 6/6/95 - D. Wolf

See changes to following functions:

FMLD, FMSV	- Framestore load and save by filename instead of xxx.FS.yyyy
GROF, GRON	- Error checked for validity.
QUIT	- Disabled for now - broken!
DOEN	- Fixed

A number of functions have been REMOVED! They are meaningless in the new Toaster/Flyer software environment (releases >3.5). The removed functions could have been potentially harmful if run. As a reminder to all you ARexx programmers to CHANGE your programs, all these functions now return an ARexx '10' error code - likely to stop your ARexx script dead so you'll notice them and change your programs! Calling these removed functions is completely harmless now, but you'll just get ARexx error returns from them anyhow.

Docs for some of these functions were correctly removed from this file some time ago, but the functions were still there and dangerous. The 27 REMOVED functions are:

PAGE, BKLD
CHGR, CKGD, RDGD, WTGD
PJLD, PJCK, PJDV, PJSV, PJDL, PJRN, PJXI, PJNM, PJMK, PJBD
FSDV, FSNM, FSMK, FSCK, FSBD, FSCT
FMDL, FMRN, FMXI
KEYP, KEYN

For example, CHGR used to let you select a different effects 'grid' - complete nonsense in the newer Toaster/Flyer environment.

****

Note about ARexx: The Video Toaster is compatible with the ARexx programming language. However, due to the complex and technical nature of programming, users of ARexx cannot be supported by NewTek Technical Support. Its inclusion is intended primarily for developers, programmers, and advanced users who are familiar and comfortable with programming concepts. Commands are subject to change without prior notice.

The Switcher Arexx interface has been implemented as a function host, rather than the more common command host interface (e.g. ToasterPaint). Arexx function hosts are initialized using the addlib() function with the name of the host or library. **The name of the switcher host is 'ToasterARexx.port' and it is case-sensitive.** ARexx messages sent to the Switcher must be in function call format. This means that **all commands must be given as arguments to the function Switcher()**, rather than being sent as direct commands to an ARexx command host. If the switcher command has any parameters, then they too must be given as arguments to Switcher() or the command will fail. **Furthermore, all parameters should be uppercase.** Unless otherwise noted, all function commands return 0 if they were sucessfully executed, and 10 if a failure occurred. **ARexx messages will only be accepted and processed when the main Switcher control panel is active.** At other times, ARexx messages will be accepted and will remain in the Switcher's ARexx message port queue until the primary switcher control panel is active, at which point they should execute (don't count on it - put the Switcher screen up!).

Most of the commands that directly impact the Switcher will also show up on the Switcher interface. Almost all ARexx commands that impact the Switcher behave exactly like their corresponding manual user interface functions from the Switcher interface. In particular, they are restricted by the active controlling effect as to whether or not their requested function will be granted. If an ARexx command function is aborted by a controlling effect, an OK return code may still be returned. In the case of a failure code, the current Switcher settings remain unchanged.

Arexx Command Set:

Switcher(AUTO)
This command will invoke an AUTO transition.

Switcher(UATO)
This command performs an un-auto, the equivalent of a Shift-Spacebar action to reverse a partially transitioned effect back off screen.

Switcher(TBAR, <TBar level>)
This command will invoke a TBar jump movement to the specified level (0-511). A failure code will be returned if the level argument is invalid.

Switcher(TAKE)
This command will invoke a TAKE.

Switcher(SLOW)
Switcher(MEDM)
Switcher(FAST)
Switcher(SVAR)
The above commands set the speed active for the transition that is currently in effect. SVAR sets the effect to the last duration chosen with the variable speed selector. You MUST make sure the effect supports variable speeds bad things can and WILL happen.

Switcher(NOPR)
This command is a NOP.

Switcher(FRES)
This command resets the frame base count to the moment this command was processed.

Switcher(WAIT, <Frame count> or GPI)
This command can wait for either a specified frame count or for a GPI trigger. If 'GPI' is the specified argument, this function will wait for a GPI trigger to occur before continuing. The trigger can be either a positive or negative trigger. In the case of GPI triggering being disabled, this function will become a NOP. If 'GPI' is not the specified argument, then the argument is assumed to be a digit field specifying a frame count. In this case the command will wait until the specified frame count from the last frame base count is reached. In the case of specified frame count already exceeded, this function will return immediately.

Switcher(CLIP, <Clip level>)
This command will invoke a Clip jump movement to the specified level (0-257). A failure code will be returned if the level argument is invalid.

Switcher(KEYM)
This command toggles the current key mode state of the Switcher.

Switcher(M001)
Switcher(M002)
Switcher(M003)
Switcher(M004)
Switcher(MDV1)
Switcher(MDV2)
Switcher(MBKG)

The above commands select the specified video source for the Main output.

Switcher(P001)
Switcher(P002)
Switcher(P003)
Switcher(P004)
Switcher(PDV1)
Switcher(PDV2)
Switcher(PBKG)

The above commands select the specified video source for the Preview output.

Switcher(O001)
Switcher(O002)
Switcher(O003)
Switcher(O004)
Switcher(ODV1)
Switcher(ODV2)
Switcher(OBKG)

The above commands select the specified video source for the Overlay output.

Switcher(GOCG)
Switcher(GOLD)
Switcher(GOSA)

These commands are now obsolete. They have been left in for compatibility reasons, but they do nothing.

Switcher(STAT,<Command>)

This command returns information about the status of the Switcher. The return value depends on which command is used. The possible commands are:

MAIN	Return status of the Program (main) row. The binary representation of the number returned will have a bit set for which input is selected on the Program row. Thus if input 1 is selected, the number returned will be 1 (binary 00000001). Similarly, input 3 would yield 8 (binary 00000100), DV2 would yield 64 (binary 00100000), as it is the sixth button on the row, and fills only the sixth bit (from the right) in the binary rep. of the returned value.
PREV	Return status of the Preview row in the same format as MAIN
OVLY	Return status of the Overlay row in the same format as MAIN
FREZ	Return status of the Freeze button: 0 for live video, ~0 for frozen video
TEXT	Return contents of popup menu in quick access panel
TBAR	Return T-Bar position (0-511)
KEYM	Return Key Mode ('BLACK', 'WHITE', or 'OFF')

CLIP Return clip level (0-257, 0 and 257 are off)

SPED Return current speed setting. The number returned is a 16-bit value whose lowest 2 bits indicate which preset speed is in effect (M=00, F=01, V=10, and S=11). The remaining 14 bits will hold the variable speed. If the 16 bit value is all 1s or 0s (i.e. -1 or 0), it means variable speeds won't work with that particular effect.

FCNT Return frame count for current effect (0-9999 frames)

KNUM Return number in quick access panel

FXNM Return the effect name for the current effect

INFO Current numeric keypad selection (SA, LD, CG).

BACK Current Background color

BORD Current Border color

TERM Current termination settings returned as a number whose lowest 4 bits represent the state of the four input terminators (input 4 is least significant bit, input1 most significant bit).

SGPI Current GPI mode (POS,NEG,OFF)

FACE Number of monitors (2 or 3)

Switcher(FMLD,volume:filename)

This command loads the contents of the frame buffer selected on the preview bus from the named volume:filename.

Switcher(FMSV,volume:filename,0)	1-field save
Switcher(FMSV,volume:filename,1)	1-field save
Switcher(FMSV,volume:filename,4)	4 field save
Switcher(FMSV,volume:filename,5)	4 field save
Switcher(FMSV,volume:filename,xxx)	4 field save
Switcher(FMSV,volume:filename)	4 field save

This command saves the contents of the frame buffer selected on the preview bus to the named volume:filename. If the last argument is 0 or 1, only 1 field of video will be saved. If the 3rd argument is NOT 0 or 1, or is absent, a standard 4-field save will occur.

Note: this will save the framestore in the format volume:filename. This command used to use a 0 to denote a 1-field save, now it uses 1 instead. Anything other than a 1 will cause a 4-field save.

Switcher(TOWB)

This command will pop the WorkBench screen up to the front and disables the Switcher/Toaster system. This function will return a failure code if WorkBench could not be opened.

Switcher(TOSW)

This command will pop the Switcher screen up to the front and enable the Switcher/Toaster system.

Switcher(QUIT)

This command will completely shut down the Switcher/Toaster system upon its arrival. All Switcher/Toaster resources will be freed and the program exited.
NOTE: This command IS BROKEN!!!

Switcher(BACK , <Index # or -1> [, <Custom BG color>])

This command will select the specified matte (background) color either from indexing the current config slice matte color FGs (0-8) or a specified custom matte color if the index is specified as -1. Custom color specifications have a valid range of 0-4096. If you want to directly specify snow, specify your custom color as a negative 16 bit word value.

Switcher(BORD , <index #>)

This command will select the specified border color via the index specified for the current config slice border color FGs (0-7).

Switcher(KOFF)
Switcher(KBLK)
Switcher(KWHT)

The above commands forces the Key mode to be off, on with black enabled, and on with white enabled respectively. Issuing this command does not guarantee that the key mode will be forced to the desired mode. If it doesn't work, the function will return a failure code.

Switcher(LVID)
Switcher(FVID)

The above commands forces the Video mode to be live or frozen respectively. Issuing this command does not guarantee that the video mode will be forced to the desired mode. If it doesn't work, the function will return a failure code.

Switcher(NOMO)

The above command does a frame capture along with motion removal.

Switcher(SGPI, <OFF| POS| NEG>)

The above command sets up the Switcher GPI trigger system. 'OFF' disables the system, 'POS' enables the system for positive GPI triggers, and 'NEG' enables the system for negative GPI triggers. The changes to the Switcher GPI trigger system made by this function WILL NOT be reflected in the HardSets file. Tech note: GPI input occurs on pin 6 of the second gameport (bit 7 of CIAA port A).

Switcher(DISW)
Switcher(ENSW)

The above commands are for disabling/enabling return of control from the ARexx environment back to the Switcher. The primary use of these commands is in running the Switcher entirely from ARexx scripts without allowing input from the Switcher interface (keyboard, mouse, etc.) to interfere with the execution of the ARexx script. Only diskchanges will still be processed.

Switcher(DIIM)
Switcher(ENIM)

The above commands are for disabling/enabling rendering, the SoftSprite system, and the mouse and keyboard IDCMP. These calls can be used to speed up the processing of ARexx switcher commands at the expense of visual feedback of the transition. When enabled, the Switcher display will be completely redrawn.

Switcher(MEMC ,'L')

The above command returns the amount of free CHIP memory or the largest CHIP chunk if the 'L' option is used. This command is ARexx dependent.

Switcher(MEMF ,'L')

The above command returns the amount of free FAST only memory or the largest FAST only chunk if the 'L' option is used. This command is ARexx dependent.

Switcher(CKTP)
Switcher(CKCG)
Switcher(CKLW)

The above commands check for the existence of ToasterPaint, CG, and LightWave respectively. They return a 1 if the program entity is currently loaded and 0 if not. These commands are ARexx dependent.

Switcher(LDTP)
Switcher(LDCG)
Switcher(LDLW)

The above commands attempt to load ToasterPaint, CG, or LightWave respectively if those program entities are not already loaded. These commands can be dangerous if the load fails and a requester comes up, hanging ARexx.

Switcher(DPTP)
Switcher(DPCG)
Switcher(DPLW)

The above commands will unload ToasterPaint, CG, or LightWave respectively if those program entities are loaded.

Switcher(STTP)

This command will enter ToasterPaint if and only if it is currently active, otherwise this command is a NOP. Once entering ToasterPaint, your ARexx message will not be returned until ToasterPaint exits back to the Switcher. So be careful that you don't hang yourself.

Switcher(OGPI)

This command causes an output GPI trigger to occur in accordance to the current settings of the GPI system. This command is a NOP if GPI triggering is disabled. Tech note: GPI output occurs on pin 9 of the second gameport (potentiometer circuit).

Switcher(TERM, <mask>)

This command sets the termination for the four video inputs. The mask will be a number from 0 to 15 whose bits (0-3) control the termination setting for inputs 1-4.

Switcher(SRGB, <filename>,<field>,<bank>)

This command saves the contents of one of the digital video buffer to a standard IFF24 type rgb file. The filename should include the complete file path. The bank argument should be 0 for DV1 and 1 for DV2. Field selects which of the 4 fields (0-3) of video stored in the frame buffer should be saved. Setting field to 4 will initiate motion removal and save 2 fields. Setting field to 5 will save a full 4-field 'color frame'. This is the best quality save.
NOTE: Not all of these settings are currently implemented .

Switcher(LRGB,<filename>,<bank>)

This command loads an RGB image into the specified digital video buffer. The filename should include the complete file path. The bank argument should be 0 for DV1 and 1 for DV2.
Note: This feature will load framestores as well.

Switcher(FACE,<monitors>)

This command sets the interface to 2 monitor or 3 monitor mode, when monitor is 2 or 3 respectively.

Switcher(GROF)

This command turns off the image hilighting system and software mouse pointer. Each call to this function MUST be matched with exactly one (1) call to the GRON function. After this call no interface elements will be hilighted. The function returns a pointer to the bitmap structure used by the screen.

NOTE: Now (>4.03) checks to see if image hilighting is ALREADY off and just returns if that is so. If GROF really happens (if image hilighting is ON when you call GROF) then you will get the pointer to the bitmap structure of the screen as a return value. If GROF doesn't happen because hilighting is ALREADY off, you will get a return value of 0 instead of the bitmap structure pointer value.

Switcher(GRON)

This command turns on the image hilighting system and software mouse pointer after a GROF. Each call to this function MUST be matched with exactly one (1) prior call to the GROF function.

NOTE: Now (>4.03) checks that image hilighting is indeed OFF before doing anything. If image hilighting is OFF when you call GRON then GRON will turn it back ON. If image hilighting is already ON when you call GRON, then it does nothing (but reports no error, either).

Toaster/Flyer Project File Format

by Steve Kell
purloined from Instinct.i - 12/8/94

```
**********************************************************************
* As of 12-6-94 the format for our Project files is as follows:
*
*    dc.b 'Project',0        ;= dc.l #'Proj' + dc.l #'ect'<<8 = File identifier
*    dc.l 0                  ;version & revision (0= 4.0)
*
*    dc.l 0,0,0,0            ;pads for future expansion
*
*    dc.l TagListSize        ;size of following System tags (including NULL)
*    TagList                 ;A Null terminated tag list of system tags
*                            ;Currently, only supporting TAG_LoadedSlices.
*
*    TBProjectEntry          ;Entry for 1st crouton in project
*    TBProjectEntry          ;Entry for 2nd crouton in project
*    etc.
*
*    0L                      ;NULL LONG to mark end of project entries
*
**********************************************************************
```

* DOC NOTE: The FG_IndexID field of a ToolBox FastGadget is entrusted to provide information * on the position of the TB FG in a ToolBox display grid as well as the sequence position of the * particular TB FastGadget in the total ToolBox FastGadget project.

* Addendum 9/18/89: The following represents a file memory image of an entry structure of a TB * FastGadget (both external and internal types) in a project script to load/save ToolBox FastGadgets * to/from disk. Has quite a bit of a direct relationship to some of the ToolBox FastGadget extensions * mentioned just above.

```
  STRUCTURE    TBProjectEntry,0

      LONG    TBPE_EntrySize    ; 0=end of file, EVEN size!!!
                                ; (all bytes upto next entry, not including these 4)

      LABEL   TBPE_EntryData

      LONG    TBPE_ObjectType
      LONG    TBPE_ObjectVersion

      LONG    TBPE_pad0
      LONG    TBPE_pad1
      LONG    TBPE_pad2
      LONG    TBPE_pad3

* could put new parameters here, if you didn't mind breaking old projects

      WORD    TBPE_FileNameSize           ;pad to an EVEN size!!!!
      LABEL   TBPE_FileName    ; Null terminated file name.
                  ; Defaults to Effects drawer, but may
                  ; contain a full path.

*     dc.b 'name here',0
```

```
*    CNOP    0,2

*    LABEL   TBPE_Parameters

* A "Special TAG list" will follow the filename. The project loader only uses the TBPE_EntrySize, *
so this data doesn't need to be a TAG list, but could be any block of memory of size *
TBPE_EntrySize-(TBPE_Parameters-TBPE_IndexID).

* Example:
*    ULONG  FGStagID_MatteColor;
*    ULONG  FGStagSZ_MatteColor;Length of MatteColor EVEN SIZE
*     UWORD.      ;MatteColor value

*    ULONG  FGStagID_FCountMode;
*    ULONG  FGStagSZ_FCountMode;Length of FCountMode EVEN SIZE
*     UWORD.      ;FCountMode value

*    ULONG  FGStagID_IndexID; the FG_IndexID for the ToolBox FG -
*    ULONG  FGStagSZ_IndexID; determines the placement of the
*     UWORD  .    ; ToolBox FastGadget

*    etc.
*    ULONG  0     ;end of tag list
```