Formula Graphics Multimedia System

16 bit Version 3.3
32 bit version 4.0
Release 95.10.22

Introduction

Multimedia Presentation System

Multimedia Elements

Background
Picture
Animation
Sound
Video
MCI
Rectangle
Text
Picture Button
Text Button
Hot Area
Hot Color
Input
Control
Remove
Debug
Hypertext
Web Document
Edit Box
List Box
Graph

Programming Language

Instruction Reference

Formula Graphics Multimedia System 3.2

Description

Formula Graphics Multimedia System will bring your artwork, sounds and animations together to create highly interactive multimedia titles. The system was designed to simplify the task of authoring on a production level, to allow the creative content of a presentation to be as rich as possible, and to make it possible to achieve an unlimited amount of interactivity.

Formula Graphics Multimedia System takes the finished artwork and animations from graphics packages such as Photoshop, Animator or 3D Studio, or from image libraries and sound recordings, and it presents this content in ways which give the best possible result under Windows.

Formula Graphics Multimedia System is very quick and easy to use. It can be used by anyone in the home, office or classroom to create very professional looking presentations and games.

Features

Presentation system: Formula Graphics Multimedia System can be used to produce any type of multimedia presentation. Presentations can be 256 color, 16 bit or 24 bit color. Using simple techniques, very high levels of interactivity can be achieved.

High speed, high performance graphics: Formula Graphics Multimedia System uses all of the latest video technology such as Video for Windows. It automatically adapts to your system and knows how to get the best performance out of any video hardware.

Dynamic palette management: Formula Graphics Multimedia System dynamically manages palette colors during 256 color presentations. Any number of backgrounds, pictures and animations with different palettes can be displayed at the same time.

Object oriented language: Formula Graphics Multimedia System high performance object oriented language has the power to explore the limits of mutimedia.

Applications

*** CDROM titles** - everything you need to produce top quality multimedia titles.

*** Multimedia education** - teach your people with hypertext and animated graphics.

*** Sales presentations** - build a slick, animated demonstration to sell your product.

*** Action and adventure games** - animated sprite based games engine.

Multimedia production

The following steps will give you some guidelines on using Formula Graphics Multimedia System to create a multimedia presentation.

1. Design the presentation. A storyboard should be developed which outlines the contents of the presentation on a screen by screen basis. Each screen would usually contain a heading, some text, a picture or two and maybe an animation.

2. Use a graphics package such as Photoshop to create the artwork. Start by designing some backgrounds which suit the theme of the presentation. Then design each screen of the presentation by laying out pictures and text over the screen's background.

3. Use Formula Graphics Multimedia System to process the artwork. Screen layouts can be loaded into the graphics window, the background can be removed by subtracting, and the remains can be cut out as picture elements. As each picture is added to the presentation, it can be reduced in colors and the subtracted area can be made transparent.

4. An animation package can be used to create 2D and 3D animations for the presentation. Animations are a great way to enhance a presentation and are particularly good for product demonstrations. Formula Graphics can be used to filter, reduce the number of colors, and convert the animations into the desired format before adding them to the presentation.

5. Sound and motion video can be recorded or captured from tape and included in the presentation. Background music, voice overs and video sequences are an important part of any presentation.

6. Other types of elements can also be included in the presentation. It is easy to compile a list of elements for each screen. The order of the elements can be changed using the up and down cursor keys. Most elements can be selected by double clicking them, then they can be resized or dragged around the screen.

7. A high degree of interactivity can be given to the presentation by using hot elements such as picture buttons, text buttons, hot areas, hot colors and hypertext. These elements can be made to carry out a list of actions when they are clicked. A simple system of logic can be used to relate elements and events.

8. Formula Graphics Multimedia System contains a high performance, object oriented language. This language may be used to process artwork for the presentation in an unlimited number of ways. It may also be included in the presentation to provide limitless interactivity. It can even be used to write game action sequences with animated sprites.

9. The final presentation can then be archived so that all of the artwork is stored in only one easily distributable file. Formula Graphics Multimedia System can then be used as a presentation player by specifying the presentation name on the command line.

Bits

The way a computer works is by using bits. A bit can be either zero or one. Groups of bits can be used to represent numbers. Bits are usually grouped together in multiples of 8. With all the possible combinations of zeros and ones, 8 bits can be used to represent any number between 0 and 255.

Bitmaps

A bitmap is a way of storing an image. The image is divided up into tiny units of color called pixels. The size of a bitmap can be given as X pixels wide and Y pixels high. The color resolution of the bitmap refers to the number of bits used to store each pixel color.

A 256 color bitmap uses 8 bits per pixel and has a corresponding table of colors called a palette. Each pixel can have a value between 0 and 255, and each value refers to the position of a color in the palette. Each color in the palette has 8 bits of red, 8 bits of green and 8 bits of blue.

A 16 bit color bitmap does not use a palette, the red, green and blue color components of each pixel are stored using 16 bits. There are two variations: RGB555 uses 5 bits of red, 5 bits of green and 5 bits of blue (32768 colors). RGB565 uses 5 bits of red, 6 bits of green and 5 bits of blue (65536 colors).

A 24 bit color bitmap has 8 bits of red, 8 bits of green and 8 bits of blue for each pixel. There are 16 million possible color variations and the smallest differences between them can barely be distinguished by the eye.

Palettes

A palette is a table of colors. There are 256 colors in a palette, and each color has 8 bits of red, 8 bits of green and 8 bits of blue. Palettes are only used with 256 color bitmaps or 256 color video modes. 256 color bitmaps are the most common types of bitmaps, and most video cards have 256 color modes.

Microsoft Windows maintains a palette called the system palette. When Windows is running in a 256 color video mode, the system palette is used by the video card. Windows reserves 20 colors in the system palette for visual elements such as windows and icons, this leaves only 236 colors which can be used for presentations.

When Formula Graphics Multimedia System is the active application, it takes control of the system palette. When a 256 color bitmap is displayed, first the colors in the bitmap's palette will be copied to the system palette, then the bitmap will be displayed using those colors.

Graphics window

Formula Graphics Multimedia System has a window called the graphics window, which can be used to display bitmaps and palettes. The size and color resolution of the graphics window can be set by choosing **Options** from the Graphics menu. The size can be specified in horizontal and vertical pixels. The color resolution can be 256 colors, 16 bits or 24 bits.

The graphics window uses the computer's memory to store the bits of color for all of its pixels. The amount of memory required depends on the width, height and color resolution of the window. A window of any size may be opened, but the operating system must have enough memory for the size you specify.

When a bitmap is displayed, its bits of color will first be copied to the memory of the graphics window, then the video display driver will be used to copy these bits of color to the video card. By using the graphics window memory as a buffer, Formula Graphics presentations can be developed independently of the video hardware they run on.

After choosing a suitable resolution, select **Open Window** from the Graphics menu and the graphics window will appear. There are a several different types of Graphics Window. Formula Graphics Multimedia System will choose the one which gives the best performance for the selected color resolution and video mode:

WinG: If the Microsoft WinG drivers are installed, then this type of 256 color window will give high performance on most video modes under Windows 3.1.

Fast 16 and 256: This type of window is very fast for a 256 color window on a 256 color video mode or a 16 bit color (RGB565) window on a 65536 color video mode under Windows 3.1.

CDS: This type of window will give the best possible performance on the Windows 95 and Windows NT operating systems.

VFW 256, 16 and 24: If Microsoft Video for Windows is available, then this type of window will provide color reduction and dithering for 256 colors on a 16 color video mode, or 16 bit (RGB555) or 24 bit colors on a 16 or 256 color video mode.

DIB 256 and 24: If no other alternatives are available, then this type of window will be used.

* WinG and Fast windows deliver the best performance under Windows 3.1. But these modes are not reliable on all video cards, and WinG is not reliable under Windows 95. These window modes must be enabled by setting the [Graphics] Fastmode switch in the "presentation.ini" file.

Mouse cursor

The position of any pixel in the graphics window can be given by the term **X,Y**. The **X** value is the pixel position across the screen and the **Y** value is the pixel position down the screen. The position of the pixel in the top left corner on the screen is 0,0.

As the mouse cursor moves across the graphics window, its position will be displayed on the status bar. If the left mouse button is clicked on a pixel, then the red, green, and blue components of the selected pixel color will be displayed on the status bar.

If the left mouse button is held down while the cursor is dragged across the graphics window, then a capture rectangle will appear over the selected area. The capture rectangle can be resized or moved around the graphics window. The capture rectangle will disappear when the left mouse button is clicked again.

A grid width value can be specified in the Graphics Options. As a capture rectangle is dragged, resized or moved, it will snap to a position in the grid. A grid width value of zero will disable this feature.

A capture rectangle can also be moved around the screen by pressing the up, down, left and right arrow cursor keys.

Loading a bitmap

After selecting **Load Bitmap** from the Graphics menu, a Load Bitmap dialog box will open. A bitmap file can be chosen. This bitmap will be displayed in the graphics window. If the color resolution of the bitmap differs from the color resolution of the graphics window, then the bitmap will be converted before it is displayed.

The **Subtract Bitmap** function can be used to remove a background image from a bitmap image. First load the bitmap file into the graphics window. Then select Subtract Bitmap from the Graphics menu and the background file can be selected from the Subtract Bitmap dialog box. The two images will be compared and all corresponding pixels with the same color will be set to the specified subtract color.

An area of the graphics window can be selected with a capture rectangle before choosing **Save Bitmap** from the Graphics menu. After choosing a bitmap file name, the selected area will be captured (converted into a bitmap) and saved. If no area was selected, then the entire contents of the graphics window will be captured and saved. If the color resolution of the chosen file format differs from the color resolution of the graphics window, then the image will be converted after it is captured.

Bitmap file formats

Bitmap files come in a variety of formats. Some formats store the image as raw data, and other formats use compression techniques to reduce the amount of space required to store the image.

The **BMP** format can either be 256 colors, 16 bit color (RGB 555) or 24 bit color. This format does not use compression. It was developed by Microsoft and is ideal for exchanging bitmaps between different applications.

GIF is a 256 color file format. It uses a technique called LZ compression which offers excellent reduction for complex images. This is the recommended format for 256 color bitmaps.

JPEG is a 24 bit color format. The JPEG compression technique achieves a high compression ratio at the cost of slow decompression and a slight loss in image quality.

Optimizing a palette

An area can be selected with a capture rectangle before choosing **Optimize Palette** from the Graphics menu. After choosing the appropriate options and pressing the Optimize button, the selected area of the graphics window will be analyzed and a new palette will be constructed using the most popular colors. If no area was selected, then the entire contents of the graphics window will be analyzed.

The number of colors to be used in the new palette can be specified, all other colors will be set to black. If the Foreground bias option is set, then the new palette will be made up of 3/4 popular colors and 1/4 wide spectrum of colors.

A bias color can be specified. Preference will be given to colors that are nearest to the bias color. Biasing away from dark colors generally produces a palette of more useful and visually appealing colors. The default settings in the color optimization dialog box will usually produce the best results.

Animations

An animation is a series of bitmaps. When the bitmaps of an animation are displayed one after the other, they give the appearance of movement. Each bitmap in an animation is called a frame, and a typical frame rate for an animation is 12 frames per second. Any slower and the motion becomes jerky, any faster and the system resources will be strained for no reason. Formula Graphics Multimedia System uses four styles of animation:

The first of these are **Simple** animations, which has one complete image for each frame.

Delta animations have a complete image for the first frame, then for each frame thereafter, only the changes between this frame and the last frame are saved. This is the traditional method of storing animations.

An **Overlay** animation is the same as a Delta animation except that the first frame of the animation contains only the changes between the first frame and some specified background image. This method can be used to save disk space by not storing the initial background.

In a **Sprite** animation, each frame contains only the changes between that frame and some specified background. As the animation is played, the background is restored before each new frame is displayed. A sprite animation can be played transparently over any background.

The performance of an animation will be a compromise between image quality and playback speed. The playback speed will depend on the compression ratio of the file format. The primary bottleneck will be the rate at which the animation can be read from the disk.

Animation file formats

The **AVI** format can have any color resolution, can use any compression technique, and it can also contain a sound track. This format is supported by the MCI (Media Control Interface) component of the Windows operating system and is the recommended file format for video.

FLC is a 256 color animation format. It uses a technique called RLL which produces moderate compression ratios and fast decompression times.

MPEG is a 24 bit color animation format. The MPEG compression technique achieves a high compression ratio at the cost of slow decompression and a slight loss in image quality.

VDO is a custom 256 color animation file format. This format uses both RLL and LZH compression. It is fully supported by the presentation system and is the recommended file format for 256 color animations.

WDO is a custom 16 bit color (RGB565) animation file format. This format uses both RLL and LZH compression. It is fully supported by the presentation system and is the recommended file format for RGB animations.

Animation playback

After selecting **Animation Play** from the Graphics menu, an Animation Playback dialog box will open. After pressing the browse button, a Load Animation dialog box will open and an animation file can be chosen. The rate at which the animation plays back can be specified.

Using single frame steps, each time the left mouse button is clicked, another frame of the animation will be played. When the right mouse button is held down, the animation will be played continuously.

The maximum frame rate will be the most number of frames that can be read from disk, decompressed, and displayed per second.

Converting an animation

After selecting **Animation Convert** from the Graphics menu, a Convert Animation dialog box will open. The Convert button will convert an animation from any file format to any other file format. The Source File(s) can either specify an animation, or a multiple selection of single bitmap files.

The mode of the destination animation can be specified, the common default mode is Delta If this mode is set to either Overlay or Sprite then a background bitmap should also be specified. If you do not specifiy one then the current image in the graphics window will be used as the background.

If the Palette button is pressed, the palette of the source animation will be loaded and displayed. If the destination animation needs a palette, then it will be created using whatever palette is currently being displayed.

Each frame of the source animation will be displayed at the specified display position. Each frame of the destination animation will be captured from the specified capture position with the specified capture size. A starting frame, finishing frame and frame step can be specified for the source file(s). If no start, finish or step is specified, then the conversion will start at frame zero and continue on steps of one until the end of the source animation.

If the Optimize button is pressed, a color Optimization dialog box will open. After the appropriate options have been selected, each frame of the source animation will be analyzed, and a new palette will be constructed using the best range of colors accross the entire source animation. Start, finish or step values may be used to limit the color sample and speed up the process.

Two Image filters are available, zero values will disable these filters. The antidither filter will test for runs of horizontal pixels with similar colors, these pixels will then be set to the same color. Antidither can be used to greatly increase the compression ratio of RLL animations. The delta filter will test for similar colors between frames.

Preparing to convert an animation

It is recommended that the size resolution of the graphics window be set to the size of the animation and the color resolution be set to 24 bit color before converting any type of animation. If only an area of the window is to be included in the new animation, then this area should be selected first using a capture rectangle so that its coordinates are automatically recorded.

It is highly recommended that all VDO and WDO animations be processed with some degree of antidither. The antidither value should be increased to the point where the loss of quality in the image is still acceptable.

Converting to a sprite animation

A sprite animation can appear to play transparently over any background. A sprite animation must first be converted to either the VDO format for 256 colors, or WDO for 65536 colors. These file formats usually only store the difference between one frame and the next, but when they are saved using the Sprite option, then each frame only stores the difference between the moving image and its background. All background information is removed.

Before you can convert to a sprite animation, you must first have a background bitmap. This bitmap must be the exact image of the area of the animation which you wish to make transparent, and it must be consistent from frame to frame. The RGB values of each pixel in the background bitmap should be the same as those in the background of the animation.

If you do not specifiy a background bitmap then the current image in the graphics window will be used. The new sprite animation will contain only the differences between each frame of the animation and this

background bitmap.

Output to a printer

The image in the graphics window can be output to a printer by selecting the **Print Window** option from the Graphics menu. A printer options dialog box will appear and you can choose the most suitable options before printing.

If the printer is set to portrait mode then the image will appear at the top of the page with proportional dimensions. If the printer is set to landscape mode then the image will be stretched to fit the entire page.

The image in the graphics window can also be printed out during a running presentation by pressing <Ctrl+P> on the keyboard. The presentation must be waiting for input from the keyboard.

Multimedia Presentation System

Presentations

A new presentation can be opened by selecting **New** from the Presentation menu. A presentation window will open and you will be asked to save the new presentation to establish a presentation name and directory.

Presentations are divided up into screens. The names of all the screens in a presentation will be listed in the presentation window. A new screen can be added to the presentation by selecting **New Screen** from the presentation menu. As each new screen is added to the presentation, its name will be added to the screen list.

Screens are made up of elements. When a screen is open, a list of its elements will appear in the presentation window. Elements are things like pictures, animations and sounds. As each new element is added to the screen, its name will be added to the element list.

A presentation can be shown by selecting **Show Presentation** from the Presentation menu. When a presentation is being shown, the screens in the screen list will be presented one after the other until they are finished. As each screen is presented, the screen's elements will be played or displayed one at a time in the order they are listed.

Starting a presentation

To begin a presentation select **New** from the Presentation menu and a presentation window will open. To open our first screen select **New Screen**. After entering a screen name, a graphics window will open. The size and color resolution of the graphics window will be those specified in the presentation options. A screen window will also open and the screen name will be added to the presentation window's screen list.

As our first element, choose **Background** from the Element menu. A Load Background dialog box will open. The background bitmap we select will be displayed in the graphics window. A Background Element window will open, and the background name will be added to the presentation window's element list.

Now we can add a picture by choosing **Picture** from the Element menu. A Load Picture dialog box will open. The picture bitmap we select will be displayed in the graphics window. A picture element window will open, and the picture name will be added to the presentation window's element list.

The picture can be selected at any time by double clicking on it in the graphics window. The selected picture can be repositioned by dragging it with the mouse.

We can add a sound by choosing **Sound** from the Element menu. After selecting a sound from the Load Sound dialog box, a sound element window will open and the sound name will be added to the presentation window's element list. We can test the sound by pressing the Play button in the element window.

Finally, we can add an input by choosing **Input** from the Element menu. After entering an element name, an input element window will open and the input name will be added to the presentation window's element list. Select the "key press / mouse click" option in the input element window.

Now that we have put together our first screen, we can test it by selecting **Test Screen** from the Presentation menu. First the background will be displayed, then the picture, and then the sound will play. When the presentation gets to the input, it will stop and wait for the user to either click on a mouse button or press any key on the keyboard.

Screens and elements

Screens and elements can be arranged in any particular order by selecting them with the mouse and moving them up or down with the cursor keys.

By double clicking on a screen name, the selected screen window will open and its elements will be displayed in the graphics window.

By double clicking on an element name, the selected element window will open. Most elements can also be selected by double clicking the element's position on the screen.

Screens and elements can be removed from the presentation by selecting them with the mouse and pressing the delete key.

Element positions

Most elements can be resized and moved by dragging them into a new position. As an element is being dragged, its position will be displayed on the status bar. The position of the element can be finely tuned by pressing the up, down, left and right cursor keys.

Choosing **Position** from the Element menu will open a position dialog box. The position of the currently selected element will be displayed. The element can be moved by changing its position values and pressing the Move button. A new position can also be entered by selecting an area with the capture rectangle before choosing Position from the Element menu.

Any changes to the position of an element can be undone by choosing **Undo** from the Element menu.

Displaying, undisplaying and removing

Elements can be displayed and undisplayed. When a picture element is displayed, the area of the screen under the picture is preserved. When the picture is undisplayed, the area under that picture is restored. It is important that elements are undisplayed in the reverse order that they were displayed, otherwise they may leave artifacts.

Elements can also be removed. When a picture element is removed, the preserved area under the picture will be discarded without being restored. When a background element is displayed over the top of other elements, these elements must be removed rather than undisplayed so that the background image is not disturbed.

After a screen is finished, the elements on the screen will be undisplayed according to the options given in the screen window.

Palette management

The presentation system dynamically manages the system palette during 256 color presentations. Before an element with a palette is displayed, any colors used by that element are allocated to free color positions in the system palette. If two elements use the same color, they will share that palette position. If no free positions are available, the nearest color will be found.

As the element is displayed, its colors will be remapped to the system palette. In the case of bitmaps, only colors that are actually used in the image are allocated positions. In the case of animations, any colors that are not used in the animation should be set to black. After a bitmap is undisplayed or removed, its palette positions will be freed.

Care should be taken to ensure that there are no more than 236 colors being used on the screen at one time. Formula Graphics Multimedia System provides excellent color optimization to reduce the number of colors in each element to the minimum possible number.

More detail

The system works by analysing each image before it is displayed, each unique palette color in the image is allocated a position in the system palette. Only colors that are used are allocated. Then the palette index of each pixel in the image is remapped as it is displayed.

If all of the available system palette positions are taken, then the system will find the next nearest available color. When an image is removed from the screen, then its palette colors are deallocated to make room for the next image.

Condition codes

There are 1000 condition codes and each one can be represented by a number between 1 and 1000. Each condition code has only two states, it can either be true or false. By setting condition codes to true or false, and by testing these codes, a high level of interactivity can be given to a presentation.

Each screen and each element has a test option in which one or more condition codes can be specified. If a single condition code is specified, then that screen or element will only be played or displayed if the condition code is true. If the condition code is a negative number, then that screen or element will only be played or displayed if the condition code is false.

For example, if an element has 20 specified as its test option, then when the presentation reaches that element, it will only display the element if condition code 20 is set to true. If -20 is specified as its test option, then it will only display the element if condition code 20 is set to false.

Several condition codes separated by the **&** (and operator) and **|** (or operator) can also be specified. The screen or element will only be presented if the overall condition is true. As an example the condition "20 & -30" will only be true if both 20 is true and 30 is false.

A condition code can be set to true by using a "set" command in an action box. For example, "set 20" will set condition code 20 to true. Two or more condition codes can be set by separating the codes with commas. For example, "set 20,30" will set both 20 and 30 to true. A range of condition codes can be set to true by specifying the lower and upper codes separated by a forward slash. For example, "set 20/30" will set all codes between 20 and 30 inclusive to true.

A condition code can be set to false using a "set" command with a negative code value. For example, "set -20" will set 20 to false. Two or more condition codes can be set to false by separating the negative codes with commas. A range of condition codes can be set to false by specifying a negative value for the first code. For example, "set -20/30" will set all codes between 20 and 30 to false.

Condition codes 1 to 12 are reserved for use by function keys, for example when the F10 key is pressed, condition code 10 will be set to true. Before a presentation begins, all condition codes will be set to false.

Action Boxes

Some elements have an action box. An action box may be used to specify commands. In most cases the commands in an action box will only be carried out after a certain event, such as the pressing of a button element or the clicking of a hot area.

The **set** command may be used to set a condition code, a number of condition codes, or a range of condition codes. As an example, "set 20" will switch condition code 20 on. Any elements that use 20 for their test values would then be displayed.

The **hit** command may be used to toggle a condition code, a number of condition codes, or a range of condition codes. The specified codes will be switched on, the screen will be updated, and then the codes will be switch off. Any element using one of these codes as a test condition will be played, displayed or redisplayed.

The **inc** command may be used to move the position of any active condition code within a given range in an upwards direction. As an example, if condition code 20 was set, then "inc 20/25" would switch 20 off and switch 21 on.

The **dec** command may be used to move the position of any active condition code within a given range in a downwards direction. As an example, if condition code 25 was set, then "dec 20/25" would switch 25 off and switch 24 on.

The **let** command may be used to assign a new value to a floating point variable or to assign a new string to a byte array. The variable name must have already been initialized in the script. As an example: let v = 7.5. As another example: let $a = "hello".

The **call** command can be used to execute a procedure in the presentation's script. The procedure name must be specified. Parameters can be passed to the procedure by first assigning values and strings using let commands. As an example: call initialize.

The **page** command is only for use with a hot word in a hypertext element or a web document. The current page will be undisplayed and the specified page will be displayed. The condition codes associated with all other pages will be switched off and the code associated with this page will be switched on. As an example: page 10.

The **play** or **display** command can be used to play or display an element on the screen. If the element is already displayed, then it will be redisplayed. If the element name has an extension or uses more than one word then it should be enclosed in inverted commas. As an example: display "picture.gif".

The **undisplay** command can be used to undisplay an element from the screen. If the element name has an extension or uses more than one word then it should be enclosed in inverted commas. As an example: undisplay "picture.gif".

The **undisplay all** command will undisplay all elements on the screen.

The **screen** command can be used to jump to another screen. If the screen name is more than one word then the screen name should be enclosed in inverted commas.
As an example: call "screen one".

The **continue** command can be used to continue past an "activate hot elements" input element.

The **goto** command can be used to jump to a specified element in the list. If the element name has an extension or uses more than one word then it should be enclosed in inverted commas.

The **backscreen** command can be used to jump back to the previous screen.

The **break** command can be used to end the current screen.

The **exit** command can be used to end the presentation.

More than one command can be given in an action box. If there is more than one command then each command can be separated by either a space, a comma, or a semi colon. The following are examples of multiple commands:

```
set -20/30,25 hit 40,41 inc 50/60
call initialize let v = 7.5 let $a = "hello" call activate
undisplay all break
```

Screen options

The FnKey option on the screen window can be used to specify a function key. If the specified function key is pressed while any other screen is being shown, then that screen will finish and the presentation will jump to this screen. The default value of zero will disable this function.

The Test option may be used to specify a condition. The Next Screen option may be used to specify the name of the next screen to be displayed after this screen.

If the "Retain elements" option is set, then when the screen finishes it will not undisplay any elements. If the "Undisplay elements" option is set then all elements will be undisplayed when the screen is finished.

If the "Undisplay changes" option is set then all elements will be undisplayed except for those elements whose names appear in element list of the next screen. If this option is chosen then no two elements on the same screen should use the same name.

If the "Right mouse advance" option is set, then the right mouse button can be clicked to advance past an input control. If the "Keyboard advance" option is set, then any key on the keyboard can be pressed to advance past an input control.

Presentation options

Selecting **Options** from the Presentation menu, a presentation dialog box will open.

When a presentation is being shown, the Presentation Title will appear on the presentation window's title bar. If the size of the presentation window is the same size as the video screen, then the presentation will be played on a full screen window with no title bar.

An optional password may be specified. You will not be able to edit the presentation without specifying the password. The presentation will always be able to be shown, but it cannot be edited without first specifiying the correct password. If the password box is empty then no password will be required.

A script may be included in a presentation. The script will be opened before the presentation begins and closed after it is finished. At any stage of the presentation, any procedure in the script may be executed. For more details, see the section on the Presentation script.

"Prompt before save" will prompt the user before saving a presentation. "Forward cursor key" enables the user to advance one screen by pressing the forward cursor key. "Backward cursor key" enables the user to go back to the previous screen.

The "Escape key to exit" option enables the escape key to exit the presentation. This option should not be used in conjunction with any script that detects mouse movements or button clicks. For action games and the like, this option should be disabled.

If the "Read file archives" option is set, files will be loaded from the presentation's archive rather than from the presentation's directory. Having this option set without a valid archive file will cause file reading errors. This option will also display old artwork from the archive rather than new artwork from the presentation directory.

The "Graphics Window" options can be used to set the size and colour resolution of a presentation. These options also specify the resolution of the graphics window which is opened when a screen name is double clicked in the presentation window screen list. For more details, see the section on the Graphics Window.

The "Cover desktop" option will blanket the whole desktop with black before opening the presentation window. The "Always on top" option will prevent any other application from being used while the presentation is running. One common use for this feature is in the development of multimedia kiosks.

Presentation script

A script can be included in a presentation. Its filename must be specified in the presentation options. The script will be opened before the presentation begins and closed after it is finished. During that time all of its data will remain valid. At any stage of the presentation, any procedure in the script may be executed.

A procedure can be executed by calling its name using a **call** command in the action details of an element. A Control element can be used to unconditionally carry out a set of actions, and this is the element that you should use to directly execute a procedure. The call command can also be used in the action box of buttons and other hot elements to execute a procedure upon the triggering of some event.

The execution of a procedure may also be multiplexed with the detection of hot element events by using an Input element with the Call option. The Call option specifies a procedure name. This procedure will be executed, then the hot elements will be scanned, then this procedure will be executed again, and so on until either a hot element is activated, or the procedure returns a zero value.

Parameters can be passed from the presentation to the script using the "let" command. The specified variables must first have been initialized in the script. More than one command can be specified in the same action box. The following example shows how to set the value of a floating point variable and a string before calling a procedure.

```
let a = 10 let $a = "dollars" call set_money
```

Scripts can be used to control almost any part of the presentation system. Scripts can also open instances of other scripts, and can call procedures in those scripts. This functionality provides the basis for object orientation.

Archiving

Choosing **Archive** from the Presentation menu will open an Archive Files dialog box. If the "New archive" button is pressed, an archive file will be created. This archive file will contain a copy of every file used in the presentation. The purpose of archiving is to minimize the number of files and provide a secure method of distributing the presentation.

After archiving, the entire contents of the presentation will be contained in only two files. The first file will be the presentation database file with the extension "dbx". The second will be the newly created archive file with the extension "db0".

A presentation may need to be changed after being archived. As an alternative to rearchiving the whole presentation, a supplementary archive file can be created containing only the changes. The "Supplementary" button can be used to create up to nine supplementary archive files. These archive files will have the extensions db1, db2, db3, etc.

Only files whose names appear in element window details will be archived. File types such as AVI, FLC and MIDI which require MCI playback will not be archived and will need to be distributed separately. Any other files that are refered to only by the presentation script can also be archived by listing their names in the "Additional files" box.

Once a presentation has been archived, it cannot be shown without its current archive file. If the archive file is no longer desired, or if it becomes corrupted, it can be cleared from the presentation by pressing the "Clear archives" button.

Distribution

After a presentation has been archived, it can then be played as a stand alone application in a window of its own. The main presentation system file is called "formula.exe". You can specify the name of any presentation in its command line, for example "formula.exe example". The specified presentation will be played without opening the presentation editor.

If the presentation files are in a different directory or on a different machine then the full path of the presentation can be specified on the command line. No file name extension should be given.

If the file "formula.exe" is renamed to "*presentation*.exe", and if the presentation files are in the same directory, then the presentation will be played. A graphics window will open in the centre of the screen, the presentation will be shown until it is finished and then the presentation system will close.

For example, the "formula.exe" file can be renamed to "example.exe" and when this file is executed, the presentation contained in "example.dbx" will be played as a stand alone application. Only when the system is called "formula.exe" will it be a presentation editor.

The final 16bit presentation can be distributed by including the following files:

presentation.exe - presentation system (formula.exe version 3.x)
presentation.dbx - presentation database file (remember to use a password)
presentation.db0 - presentation archive file
formsync.dll - used for timing and synchronization

The final 32bit presentation can be distributed by including the following files:

presentation.exe - presentation system (formula.exe version 4.x)
presentation.dbx - presentation database file (remember to use a password)
presentation.db0 - presentation archive file

Files such as AVI, FLC or MID which require MCI playback must also be included separately along with any drivers that may be required. The Formula Graphics Multimedia System can also be used to develop a setup utility to decompress and install all of your files, drivers and icons.

The "msvideo.dll" file is a distributable component of Microsoft Video for Windows which allows a 256 color presentation to be shown on a 16 color video card, or a 24 bit color presentation to be shown on a 16 or 256 color video card. The file is not necessary, but if it exists, it will be used. You should only install this file with Windows 3.1, Windows 95 already has Video for Windows.

Multimedia Elements

Background

After selecting Background from the element menu, a Load Background dialog box will open and a bitmap file can be selected as a background. The background will appear in the graphics window and a background element window will open with some options.

When a background is displayed, in most cases it will be displayed over the top of other elements on the screen. If the Remove All option is activated, then all other elements on the screen will be removed after the background has been displayed. In some cases, the background will not cover the entire screen and you will not wish to remove all other elements. Any background that was previously displayed will always be removed.

A rectangle element can also be used as a background.

To prepare a background

If the number of colors in the background needs to be reduced, then display the background in the graphics window using Load Bitmap, reduce the number colors using Optimize Palette, and save the bitmap as the background file. Then load the background file as a new background element.

Picture

After selecting Picture from the element menu, a Load Picture dialog box will open and a bitmap file can be selected. The picture will appear in the graphics window and a picture element window will open with some options.

If an area was selected with a capture rectangle before choosing Picture, then a Capture Picture dialog box will open. After specifying a file name, the selected area of the graphics window will be captured and saved. This new bitmap will become our picture element.

After pressing the Transparency Choose button, the left mouse button can be clicked on any part of the graphics window to select a transparency color. We can cancel our selection by clicking the right mouse button. We can keep selecting colors until a suitable transparency has been found. The transparency function can be enabled and disabled.

To prepare a picture

Design a screen layout using a paint program. This layout would usually consist of pictures and text placed over the top of a background. Display the screen layout in the graphics window using Load Bitmap. Use Subtract Bitmap to subtract the background, choose a subtract color which is different to any other color in the layout. The number of colors in the layout can be reduced if necessary.

The individual images in the layout can now be added to the presentation as picture elements. Place a capture rectangle over each image and select Picture from the Element menu. The selected area will be saved as the picture bitmap.

After all the pictures have been captured select Display All from the Presentation menu. Then for each picture element choose a transparency to hide the subtract color.

Animation

After selecting Animation from the Element menu, a Load Animation dialog box will open and an animation file can be selected. The first frame of the animation will appear in the graphics window and an animation element window will open with some options.

We can set the playback rate of the animation, the default rate is 12 frames per second. If the playback rate is set to zero then the animation will be played at the maximum possible rate.

If the Skip option is set then the animation will skip frames if necessary, to achieve the specified frame rate. Frame skipping does not look good with delta animations.

If an non zero value is specified in the Series option, and two animations are played on the same screen with the same series value, then the first animation will removed from the screen before the second animation plays. The transition will not be noticeable.

The Key Frames option can be used to set key frames during the playing of an animation. After each key frame is played, the next element in the element list will be activated. Up to 32 key frames can be specified.

For example, if the key frames "10,20,30,40" are specified, then after tenth frame the first element after the animation will be activated, after the twentieth frame the second element after the animation will be activated, and so on until the fortieth frame. If the continue option is set then the animation will always play until it is finished.

If any key frame number is greater than the frames in the animation, then the animation will be repeated until the specified number of frames have been played. If the Repeat option is enabled then the animation will continue to repeat until a key is pressed on the keyboard.

To prepare an animation

For the best performance, the animation should first be converted to either the VDO for 256 colors or the WDO file format for 16 and 24 bit color. This can be done by selecting Convert Animation from the graphics menu. Only the VDO and WDO file formats have palette remapping and sprite mode transparency.

If the destination file format is 256 colors then a palette will need to be established. Either use the Palette button in the Animation Conversion window to load the palette from the source animation or use the Optimize button to analyse the source animation and create the optimum palette.

The destination animation can be saved as a simple, delta, overlay or sprite animation. The mode chosen will depend on how the animation is used. If the animation needs to appear transparently over the background, then the sprite mode should be used.

As the animation is converted, it can be hammered with the antidither filter to increase the compression ratio. The antidither value should be as high as possible while the loss of image quality is still acceptable.

The final destination animation can then be added to the presentation.

Playing AVI and FLC animation formats

Before you can play an AVI or FLC animation element, you must first make sure that the correct MCI drivers have been installed. If the drivers are not properly installed then the animations wont play and an error will occur.

Because of advanced drivers, operating system support and possible hardware support, there may be

performance advantages in using these animation formats.

Sound

After selecting Sound from the element menu, a Load Sound dialog box will open and a sound file can be selected. A sound element window will open with some options.

If the Play button is pressed the sound will play until either the Stop button is pressed or the sound finishes. While the sound is playing the Set button can be pressed to capture moments in time.

The Timing box can be used to set moments in time during the playing of a sound. As each moment passes, the next element in the element list will be displayed. Up to 32 moments in time can be specified.

For example, if the timings "2,4,6,8" are specified, then after two seconds of sound the first element after the sound will be activated, after four seconds the second element after the sound will be activated, and so on until after the eighth second the remaining elements on the list will be activated.

A negative value can also be included in the list of moments. A negative value will indicate a number of elements to display without any delay. After displaying this number of elements, the next element will only be displayed at the next specified moment of time.

If the Wait option is enabled then the sound will finish playing before the next element is displayed.

Video

After selecting Video from the element menu, first select an animation from the Load Animation dialog box, then select a sound from the Load Sound dialog box. The animation can be of any format, but VDO and WDO animations are recommended. The first frame of the animation will appear in the graphics window and a video element window will open with some options.

The playback rate of the animation can be specified, the default rate is 12 frames per second. If the playback rate is set to zero then the animation will be played at the maximum possible rate.

The animation will be synchronized to the sample rate of the sound. The Sync option can be used to specify a time delay between the animation and the sound. If the specified value is positive, then that number of animation frames will be played before the sound begins. If the specified value is negative, then that number of sound periods will play before the animation begins.

If the Skip option is set then the video will skip frames if necessary, to achieve the specified frame rate. Frame skipping does not look good with delta animations.

Sound files should be limited to mono 8 bit samples to reduce memory consumption, for faster performance, and because some older sound card drivers cannot distinguish between bytes per second and samples per second and this may cause problems with syncronization.

MCI

MCI (Media Control Interface) is a component of the Microsoft Windows operating system which allows easy control of multimedia devices such as sound and video players. For more information on MCI please refer to the Microsoft Windows multimedia documention.

if an area is selected with a capture rectangle before choosing MCI from the element menu, then a Load Video dialog box will open and a video file can be selected. The first frame of the video will appear at the selected position in the graphics window and an MCI element window will open.

The video file must be an AVI, FLC or other format for which the MCI drivers are installed. A video file can contain a sound track which will be played synchronously with the animation. Some types of video file can be played full screen at 320x240 resolution.

Because of operating system limitations, only one MCI video can be shown at a time. If your animation file does not contain sound, then for the best performance, convert it to the VDO or WDO format and use an Animation element.

If no area was selected then an MCI element window will open and the Command string option will be selected. A list of MCI commands can be entered in the given box. When the element is played, the specified list of commands will be carried out.

If the MIDI option is chosen, then the name of the element should be same as that of a MIDI file in the presentation directory. When the element is played, the MIDI file will be played by the MCI system. If no MIDI player exists then the element will be ignored. If the Wait option is set then the presentation will wait until the MIDI file has finished playing.

Playing AVI animations

Before you can play an AVI animation, you must first make sure that the correct MCI drivers have been installed. If the drivers are not properly installed then the animations wont play and an error will occur. The base AVI drivers are already contained in Windows 95. If you are using an advanced compression driver then this must be installed.

A Formula Graphics script can be written to install an MCI driver. The driver must be copied to the Windows system directory, the MCI sections of the "win.ini" and "system.ini" files must be changed, and then Windows must be restarted for the changes to take effect.

Rectangle

An area can be selected with a capture rectangle before choosing Rectangle from the element menu. After specifying an element name, a Rectangle dialog box will open with several options.

If the Filled rectangle option is set, then the area of the rectangle will be filled with the rectangle's colors. If the Horizontal gradient option is set, then the rectangle will be filled with a range of colors from Color 1 at the top to Color 2 at the bottom. If the 50% translucent option is set, then only every second pixel of the rectangle's area will be filled. The rectangle can also be drawn using the Radial wipe option.

If no area was selected before choosing Rectangle, then the Background option will be set and the element will be assumed to be a background. When the element is displayed, any other elements which were on the screen before it will be removed.

If the Border option is set, then the rectangle will have a border. This border can be used on the rectangle or it can be used to surround any other element. The border can have any pixel width and can be chosen to appear as a raised bevel or a sunk bevel.

Text

An area should be selected with a capture rectangle before choosing Text from the element menu. After specifying an element name, a Text dialog box will open. Up 256 characters of text can be entered. The font type and color can be chosen. An optional drop shadow can also be specified with a shadow color and a pixel offset.

The text may be selected again by double clicking its position on the screen. It can then be resized or dragged to a new position. Press the Display button to display the text.

The characters which are displayed by the element can be changed during a presentation using a "set textbox" instruction in the presentation script.

Picture Button

After selecting Picture Button from the element menu, first select an up state bitmap from the Load Button dialog box, then select a down state bitmap from the Load Down State dialog box. The down state bitmap is optional and can be disabled by pressing the cancel button. The button will appear in the graphics window and a button element window will open with some options.

If an area was selected with a capture rectangle before choosing Picture Button from the element menu, then a Capture Button dialog box will open. After specifying a file name, the selected area of the graphics window will be captured and saved. This new bitmap will become the up state of the button element. By pressing Capture in the button element window, a down state bitmap can be captured from the same coordinates.

If a non zero value is specified in the Bank option, and other button elements have the same bank value, then these buttons will be grouped together. If one of these buttons is pressed then it will stay in the down state until another button in the same group is pressed. Only one of these buttons can be in the down state at any one time.

If a condition code value is specified alongside the bank value, and if the condition is true when the button is displayed, then the button will be displayed in the down state. When the button is pressed, this condition code will be set to true, and the condition codes of all other buttons in the same bank will be set to false.

As an example, a project displays a bank of buttons and one button is selected, then the button bank is undisplayed. If the bank of buttons is displayed again, then the down state of the selected button will be preserved by the condition code.

When the button is pressed, the commands given in the action box will be carried out.

To prepare buttons

Design two screen layouts. The first with all the buttons in the up state and the second with all the buttons in the down state. Display the first screen layout in the graphics window. Then one by one place capture rectangles over the individual buttons and select Picture Button. As each button element is added to the presentation, its selected area will be captured and saved as the button up state bitmap.

Display the down state layout in the graphics window. Then for each button element press the down state capture button. The down state bitmaps will be captured from the same coordinates as the up states.

For each button enter an appropriate command in the action box. As an example, we have two buttons and a picture, when one button is pressed the picture will be displayed, when the other button is pressed the picture will be undisplayed. For our example we will use condition code 20. In the action box of the first button type "set 20", in the action box of the second button type "set -20". In the test box of the picture element type "20".

For a full list of action box commands, see Action Boxes.

Text Button

An area should be selected with a capture rectangle before choosing Text Button from the element menu. After specifying an element name a Text Button dialog box will open. Text can be specified to be displayed on the button. The font type, font color and button color can be chosen. The Bank and Code options are the same as those described in a Picture Button element.

The text button may be resized or dragged to a new position on the screen. When the text button is pressed using the left mouse button then the commands given in the action box will be carried out.

Hot Area

An area should be selected with a capture rectangle before choosing Hot Area from the element menu. After specifying an element name a Hot Area dialog box will open. A different cursor can be chosen for when the mouse is over the hot area. When the left mouse button is clicked over this hot area then the commands given in the action box will be carried out.

Hot Color

After selecting Hot Color from the element menu and specifying a name, a Hot Color element window will open. After pressing the Choose button, any color on the screen may be chosen. When the left mouse button is clicked over this hot color then the commands given in the action box will be carried out.

Input

After selecting Input from the element menu and specifying a name, an Input Element window will open. The default type of input is an "Activate hot elements".

When an "Activate hot elements" input is activated, the presentation system will stop and wait for some response from the user. If the left mouse button is pressed over a hot element (button, text button, hot area, or hot color), then the action specified in that element's action box will be carried out.

If the action results in a change in the condition codes, the presentation system will search back up to the top of the element list, removing any element whose test condition is no longer valid. The presentation system will then search from the top of the element list down to the input element, activating any element whose test conditon has just become valid.

If the right mouse button is pressed and the "Right mouse advance" option is set in the screen window, or if a keyboard key is pressed and the "Keyboard advance" option is set,
then the presentation system will advance to the next element.

If a "Key press or mouse click" input is activated then regardless of any options, the presentation system will wait until a keyboard key or mouse button is pressed before advancing to the next element.

If a "Wait for sound to finish" input is activated then the presentation system will wait until the currently playing sound is finished before advancing to the next element.

If the input type is set to "Wait ... 1/100 seconds" then the presentation system will wait until the specified number of hundredths of a second have passed before advancing to the next element.

The "Active wait ... seconds" option is almost the same as the "Activate hot elements" option. If no key or mouse button has been pressed for the specified number of seconds, then the presentation will continue past this element.

If a "Call ..." input is activated then the presentation system will perform an "activate hot elements" while simultaneously executing a procedure in the presentation script. The system will continue to test the state of the hot elements and execute the script until either the right mouse button is pressed, a key on the keyboard is pressed, or the script returns a zero value.

The procedure will be passed 4 values which must be taken by four variables. These values represent message information used by Microsoft Windows. For more details on these values, see the peekmessage command in the programming language section.

Control

A control element can accept any number of condition codes in its Test option. The condition codes can be separated by either AND (&) or OR (|) operators. For example, "20 & 30" will only enable the element if both condition codes 20 and 30 are true. "20 | 30" will enable the element if either condition codes 20 or 30 is true. AND has a higher order of precedence than OR.

If no test condition was specified, or if the result of the specified condition is true then the commands given in the action box will be carried out.

Using a Control element

One of the most common uses of a control element is to set a condition code. Another common use is to call a procedure in the presentation script. No test option needs to be specified, simply type **call** *"procedure_name"* in the action box. After the procedure has finished executing and returns, then the next element will be displayed.

Remove

After selecting Remove from the element menu and specifying a name, a Remove Element window will open. The remove element has four different options. The Remove option will remove all of the listed elements from the screen without undisplaying them (without restoring the image beneath them). The Undisplay option will undisplay all of the listed elements.

Any number of element names can be listed, one element on each line. The '*' character can be used as a wildcard, for example, if the name "ex*" is specified, then all elements whose names begin with the letters "ex" will be removed.

The "Remove all" option will remove all elements except for the current background without undisplaying them. The "Undisplay all" option will undisplay all elements except for the current background.

Debug

After selecting Debug from the element menu and specifying a name, a Debug Element window will open. When a Debug element is activated, a message box will display a number for each condition code that is true.

The Range option can be used to specify particular codes or ranges of codes. Only those true condition codes in the combined range will be displayed.

Hypertext

An area can be selected with a capture rectangle before choosing Hypertext from the element menu. A Load Hypertext dialog box will open and a Rich Text File (RTF) can be selected. The formatted text will appear in the graphics window and a hypertext element window will open.

The RTF file format is the industry standard for formatted text. Most word processors, including Microsoft Word, are compatible with the RTF format. You can type your document into a word processor and save it as an RTF file, or you can type your document into the Formula Graphics hypertext editor. Only the basic formatting features are supported: font type, font height, bold, italic, text color, line breaks and page breaks.

A Page number can be specified in the element options. This is the page of the document which will first be displayed. The default page number of zero refers to the first page in the document. This number should be left as zero if the document only has one page.

One way of displaying a multiple page document is to use a different Hypertext element for each page. This way each page can have a different position on the screen and a different position in the list of elements.

A more efficient way to display a multiple page document is to use one Hypertext element and to turn the pages as required. There are several ways to turn the page, one of the simplest is to use the following instruction in the presentation script:

set hypertext *"element"* to *page*

If the Use condition codes option is set, then each page of the document can have a condition code associated with it. The Start value will be number of the code associated with the first page. The Range value will be the number of codes to be used and this should be the same as the number of pages in the document.

As an example, suppose you have a 100 page document. The Code option could be set to the beginning of a range of unused codes, say 500. The Range option would be set to 100 to indicate 100 pages. So then the range of condition codes between 500 and 600 would be associated with the corresponding pages 0 to 100 in the document.

The advantage of using condition codes is that the page can be turned by setting condition code for the new page. Pages can be turned forward and backward by buttons using the "inc" and "dec" action commands, or by using the 'set' command.

Regardless of how the page is turned, the system will turn off all other condition codes, and turn on the one associated with the current page. A picture, sound or other element can be associated with a page by using the same condition code in its Test option.

Hot words and hyperlinks

Hypertext has the ability to specify hot words which will carry out actions when they are clicked. To specify a hot word in the Formula Graphics editor, highlight the word and select Hyperlink from the Format section of the Editor menu. To specify a hot word in a word processor, the selected words must first be highlighted using the strikethrough font with a suitable color. The action to be carried out can then be specified directly after using hidden characters.

The action associated with a hot word can be any action box command or combination of action box commands. For instance, the action may be to display another element, play a sound, jump to another screen, or execute a procedure in a script. The action may also be a jump to another page in the document. If a specified action is not a valid command then it will assumed to be a hyperlink name.

For a full list of the commands which can be associated with hot words, see Action Boxes.

Hyperlinks

At the top of each page of your document, you have the option of specifying a topic name, a hyperlink, and some keywords. To specify these items in the Formula Graphics hypertext editor, select Topic from the Format section of the Editor menu. To specify these items in a word processor, use a $ footnote for the topic name, a # footnote for the hyperlink, and a K footnote for the keywords, in that order. Separate the keywords with semicolons.

A hot word can specify a hyperlink name instead of an action command. When a hot word with a hyperlink name is clicked, then any page of the document with that hyperlink name at the top will be displayed. Any number of hot words can point to the same page, but each page must have a unique hyperlink name. The hyperlink name must be only one word comprised of letters, numbers and underscores.

Searching for keywords

A word or keyword search can be performed on a hypertext document. The following instruction will open a Text Search dialog box. Every keyword in the document will be listed. If one of these keywords is selected, then the page associated with it will be displayed. If more than one page specifies that keyword, then another dialog box will open and the page numbers or the topics associated with those pages will be listed.

search hypertext *"element"*

If any word phrase is typed into the Text Search dialog box and the Search button is pressed, then the document will be scanned for any case insensitive occurrences of that phrase. If more than one page has an occurrence of the phrase, then another dialog box will open and the page numbers or topics associated with those pages will be listed.

This method of specifying and searching topics, hyperlinks and keywords is exactly the same as that used by the Windows Help system. The same RTF files can be used by both Formula Graphics and the Microsoft Windows Help compiler. For more information, refer to the Microsoft documentation for authoring Windows Help files

Web Document

An area can be selected with a capture rectangle before choosing Web Document from the element menu. A Load Default file dialog box will open and a Hypertext Markup Language (HTML) file can be selected. The marked up text will appear in the graphics window and a Web Document element window will open.

HTML is the standard document file format for the World Wide Web. Formula Graphics has its own editor. The features supported are headings, bold, italic, line breaks, page breaks and internal hyperlinks. Formula Graphics does not support vertical scrolling, the <HR> command is used as a page break to divide a document into viewable segments.

If the computer has a connection to the internet through a modem or a network, then this element will download a document from a world wide web server and display it. The Default file will be displayed when the presentation is being edited, or when no internet connection is available.

The Server name must be a domain name such as "www.magna.com.au" and the Document path should be of the form "/~formula/formula.htm". The presentation will be held up while the server is being contacted. The delay may be considerable if the connection has a problem.

Formula Graphics only supports the essential subset of HTML commands. No other web resources are implemented in this version. The default font type and text color of the document can be specified in the element options.

A Page number can also be specified in the element options. This is the page of the document which will first be displayed. The default page number of zero refers to the first page in the document. This number should be left as zero if the document only has one page.

One way of displaying a multiple page document is to use a different Web element for each page. This way each page can have a different position on the screen and a different position in the list of elements.

A more efficient way to display a multiple page document is to use one Web element and to turn the pages as required. There are several ways to turn the page, one of the simplest is to use the following instruction in the presentation script:

set webdoc *"element"* to *page*

If the Use condition codes option is set, then each page of the document can have a condition code associated with it. The Start value will be number of the code associated with the first page. The Range value will be the number of codes to be used and this should be the same as the number of pages in the document. For more details about using condition codes to display multipage documents, see the Hypertext element.

Hot words and hyperlinks

Web documents have the ability to specify hot words which will carry out actions when they are clicked. To specify a hot word in the Formula Graphics web editor, highlight the word and select Hyperlink from the Format section of the Editor menu.

The action associated with a hot word can be any action box command or combination of action box commands. For instance, the action may be to display another element, play a sound, jump to another screen, or execute a procedure in a script. The action may also be a jump to another page in the same document. If a specified action is not a valid command then it will assumed to be a hyperlink name.

For a full list of the commands which can be associated with hot words, see Action Boxes.

Hyperlinks

At the top of each page of your document, you have the option of specifying a hyperlink name. To insert a hyperlink name at the top of a page in the web editor, select Topic from the Format section of the Editor menu.

A hot word can specify a hyperlink name instead of an action command. When a hot word with a hyperlink name is clicked, then any page of the document with that hyperlink name at the top will be displayed. Any number of hot words can point to the same page, but each page must have a unique hyperlink name. The hyperlink name must be only one word comprised of letters, numbers and underscores.

Edit Box

An area should be selected with a capture rectangle before choosing Edit Box from the element menu. After specifying an element name, an edit box will be displayed and an Edit Box element window will open with some options.

The text in the edit box may be "read only" or "upper case". The edit box may be used for one line only, or it can be used for multiple lines with optional word wrap and a vertical scroll bar. The font type, text and background colors can also be chosen.

The contents of the edit box may be set using a set editbox instruction in the presentation script. The contents can be read using a getedit instruction.

set textbox *"element"* = *"string"*

For more details, see the example on programming an Edit box.

List Box

An area should be selected with a capture rectangle before choosing List Box from the element menu. After specifying an element name, a list box will be displayed and an List Box element window will open with some options.

The font type, text color and background color can be chosen. The list box can have a thin black border. The scrollbar can be hidden when the list is less than one page long. The list box can have multiple columns and can take multiple selections. The contents of the list box can be sorted in alphabetical order.

If the Notify option is set and a procedure name is given, then the procedure will be called if any item in the list box is selected. The procedure will be passed two values. If the first value is equal to LBN_SELCHANGE then the item was clicked, if it is equal to LBN_DBLCLK then the item was double clicked. The second value will be the index number of the item.

Several instructions are available for dealing with List boxes:

listbox *"element"* add *"item"*
state = listbox *"element"* position *x*
set listbox *"element"* position *x* = *state*
reset listbox *"element"*

Using a list box

After the List Box element has been displayed on the screen, a Control element should be used to call to a procedure in the presentation script. This procedure should add a number or items to the list. When an Input element with "activate hot elements" is running, the list box can be used.

The notify option can be used to give an instant response to a click or double click on any item in the list. A procedure name must be specified, this procedure will be called when the list box is clicked. Otherwise another procedure can be called later which detects the selection state of every item in the list.

For more details, see the example on programming a List box.

Graph

An area should be selected with a capture rectangle before choosing Graph from the element menu. After specifying an element name, a Graph element window will open with some options.

The graph may be either line, bar, or pie. Graduations can be displayed in a selected font. Percentage differences can be shown between series on a bar graph. Line and bar graphs can be drawn using stacked series. The "Line width" option will not only set the width of lines, but will also set the distance between bars.

The Call option must be used to specify the name of a procedure in the presentation script. When the graph element is activated, the specified procedure will be called and will be expected to return a list of parameters. This information will be used to draw the graph.

The first parameter must be an array. This can be a one dimensional array containing the data values for each category of the graph. The size of the array will indicate the number of categories. Two dimensional arrays can be used for multiple series line or bar graphs. The size of the second dimension will indicate the number of series.

The second parameter will be a two dimensional array, containing the red, green and blue components of the graph's colors. The first color will be used for graduations, the second color will be the color of the first data series, the third color will be the color of the second data series and so on.

The third parameter will be the minimum value of the graph data, the fourth parameter will be the maximum value of the graph data, and the fifth parameter will be the increment of the graduations between the two.

Programming Language

Basics

The Formula Graphics programming language is similar to other programming languages like BASIC and C. A wide variety of instructions are available and each instruction performs some particular operation. A program can be written using a list of instruction statements. Only one instruction can be written on each line. The instructions will be executed one at a time in the order they are listed. After the last instruction has been executed, the program will end.

first instruction
second instruction
...
last instruction

There are a number of instructions available which can change the flow of execution. These include condition statements, which only execute a branch of the program if a certain condition is true, loop statements, which execute the same group of instructions again and again, and call statements, which branch off and execute some other group of instructions before returning.

Indentation is used in Formula Graphics programs to indicate the possible paths of execution. Indentation refers to the number of tabs or spaces at the beginning of a line. For example, an "if" statement can be used to test if a certain condition is true. If the condition is true, then all following statements with a greater level of indentation will be executed. If the condition is false, then those statements with greater indentation will be skipped.

if condition is true
    then execute these indented statements
    ...
in any case execute these
...

Comments can be added to a program. These are usually just a few words describing what the program does. Anything written after a *//* specifier will be considered a comment and will be ignored by Formula Graphics. Blank lines and comments can be included between indented lines without effecting the levels of indentation.

// This is a comment

Floating point variables

Floating point numbers are used in Formula Graphics programs. A floating point number can be whole number or a fraction or it can be positive or negative. When using a floating point number, fractions less than one must have a zero before the decimal point. 0.05 and -1.414214 are examples of floating point numbers:

A floating point number can also have a power of ten exponent which is specified with a lower case "e". As an example, the value of 1e6 will be 1 times 10 to the power of 6 (or 1000000). The exponent may be positive or negative.

Variables are used extensively in programming, in fact they are one of the most important ingredients in any program. There are several different types of variables. Each variable has its own individual name. A variable name can be made up of case sensitive letters, numbers and underscores. The first character must be a letter.

The simplest form of variable is a floating point variable. A floating point variable is created when a variable name is made equal to a floating point value. That variable name can then be used as a substitute for the actual value anywhere else is the program.

variable name **=** *floating point value*

A floating point value can be just a floating point number, or it can be an expression containing numbers and variables combined with operators like '+' and '-'. A wide range of mathematical operators and functions are available for use in floating point expressions and spaces can be included where necessary.

As an example consider the instruction "x = 1". The variable "x" will be assigned with the value "1". The variable "x" can then be used elsewhere in the program to represent the value "1". The value of the variable "x" can be changed by reassigning it with another value. For instance "x = x + 1" will add one to the value of "x".

Any number of variables can be assigned with values in the same instruction. The variables and values must be separated by commas, as an example "x, y, z = 10, 20, 30". A variable assignment can be written in the general form:

variable 1, *variable 2*, **...** = *expression 1*, *expression 2*, **...**

Condition statements

If the condition given in an "if" statement is true, then all following statements with a greater level of indentation will be executed. If the condition is false, then those statements with greater indentation will be skipped.

if *condition*

If an **else** statement follows an **if** statement, and the condition is false, then all statements following the **else** statement with a greater level of indentation will be executed, otherwise they will be skipped. Any number of **else if** statements can also follow an if statement.

else if *condition*
else

If more than one level of indentation is used, then each **else** statement will correspond to the **if** statement above it that has the same level of indentation. Single variable assignments can also be carried using combined condition statements:

if *condition* **then** *variable* **=** *expression*
else if *condition* **then** *variable* **=** *expression*
else *variable* **=** *expression*

The condition specified by an **if** statement can be any floating point expression. If the value of the expression is not equal to zero, then the condition will be true. If value of the expression is equal to zero, then the condition will be false. Several mathematical operators are available for comparing floating point values.

The **==** (equal) operator will compare two values, and if they are equal the result will be one, otherwise the result will be zero. Using the **!=** (not equal) operator, if the two values are not equal the result will be one, otherwise the result will be zero. The **<, >, <=** and **>=** (magnitude) operators can also be used to compare two values.

The result of two or more comparisons can be combined using boolean logic. The **&&** (and) operator will compare two values, and only if they are both not equal to zero will the result be one, otherwise the result will be zero. The **||** (or) operator will compare two values, and if either of them is not equal to zero then the result will be one, otherwise the result will be zero.

if the **!** (not) operator is applied to a zero value the result will be one, if it is applied to a non zero value the result will be zero.

Loops

A "for" loop can be used to increment the value of a variable from a starting value to a finishing value. All following statements with a greater level of indentation will be included in the loop.

for *variable* **=** *starting value* **to** *finishing value* { **step** *increment* }

The given variable will be assigned with the starting value, then all following statements with greater indentation will be executed. The value of the variable is then incremented by the optional step value. If no step value is specified, then the variable will be incremented by one.

As long as the value of the variable is less than or equal to the finishing value, then the variable will continue to be incremented, and all following statements with greater indentation will continue to be executed. As an example:

```
for n = 0 to 9
        // These lines be executed 10 times
        ...
// The program will continue here
...
```

A negative step value may be specified, in which case the loop variable will continue to be decremented for as long as its value is greater than or equal to the finishing value.

for *var1, var2, ...* **=** *val1, val2, ...* **to** *val1, val2, ...* { **step** *inc1, inc2, ...* }

Up to three variables can be specified in a single "for" statement. The variables will be incremented in separate nested loops with the first variable being in the outer loop. Using more than one loop, an area or a volume of values can be generated.

A "while" loop can be used to continuously execute all following statements with greater indentation for as long as the value of the specified expression is true (not equal to zero).

while *expression*

The following example shows a while loop used to count to 10:

```
n = 0
while n < 10
        n = n + 1
```

If a **break** statement is found during the execution of a loop, the loop will immediately finish, and execution will jump to the first line after the loop.

If a **continue** statement is found during the execution of a loop, the current iteration of the loop will finish, and the next iteration will begin again at the top of the loop.

There are some other, more specialized types of loops which will be discussed later.

Arrays

Arrays are used by many of the instructions in Formula Graphics. An array can be allocated with any number of dimensions. Each dimension can have any number of elements. Each element in an array can be used to store a value.

As well as arrays of floating point numbers, Formula Graphics also handles arrays of bytes and words. A byte is an 8 bit unsigned value between 0 and 255. A word is a 16 bit signed value between -32768 and 32767. Byte arrays and word arrays can be used to store strings, bitmaps, palettes and sounds.

Each of the following instructions will allocate an array, the elements of the array will be set to zero, and the array will be assigned to the variable:

variable **= new byte [** *dim 1* **][** *dim 2* **] ...**
variable **= new word [** *dim 1* **][** *dim 2* **] ...**
variable **= new float [** *dim 1* **][** *dim 2* **] ...**

The position of an element in an array can be specified using an index value for each dimension. An element can be assigned with a value, and then the element can be used to represent that value in an expression. Byte and word values are automatically converted to and from floating point values.

variable **[** *index 1* **][** *index 2* **] ... =** *expression*

The term *variable* **[** *index 1* **][** *index 2* **] ...** can also be used in any floating point expression to find out the value of an element.

variable **[** *index 1* **][** *index 2* **] ... =** *expression, expression, ...*

Any number of consecutive array elements can be assigned with values in the same instruction.
Only the index values of the first element need to be specified. The following example allocates a floating point array with three elements, and then assigns a value to each of the three elements:

```
array_one = new float[3]
array_one[0] = 1,2,3
```

The following example allocates a two dimensional floating point array. There are five elements in each dimension giving a total of 25 elements. Each element is then assigned a value:

```
array_two = new float[5][5]
for n,m = 0,0 to 4,4
    array_two[n][m] = n+m
```

Bitmap arrays

A 256 color bitmap can be stored in a two dimensional byte array. The size of the first dimension will equal the vertical height of the image in pixels. The size of the second dimension will equal the width of the image.

```
bitmap8_array = new byte [height][width]
```

A 16 bit color bitmap can be stored as a two dimensional word array. The size of the first dimension will equal the vertical height of the image in pixels. The size of the second dimension will equal the width of the image.

```
bitmap16_array = new word [height][width]
```

A 24 bit color bitmap can be stored in a three dimensional byte array. The size of the first dimension will equal the vertical height of the image in pixels. The size of the second dimension will equal the width of the image. The third dimension will contain red, green and blue values.

bitmap24_array = new byte [height][width][3]

Instructions are available for loading, saving, capturing and displaying a bitmap as an array. Many other instructions take bitmap arrays as parameters.

Palette arrays

A 256 color palette can be stored in a two dimensional byte array. The first dimension will specify the color index, and the second dimension will contain the red, green and blue values.

palette_array = new byte [256][3]

Instructions are available for capturing and displaying a palette as an array. Many instructions can take optional palette arrays with bitmap arrays. Instructions are also available for optimizing a palette for a bitmap, and remapping the palette indexes of a bitmap.

Sound arrays

A mono 8 bit sound can be stored in a byte array. A stereo 8 bit sound can be stored in a two dimensional byte array. The first dimension specifies the sample number and the second dimension gives the left and right channels. Each element of an 8 bit sound array will be an unsigned sample value with 128 indicating zero output.

mono8_array = new byte [sample length]
stereo8_array = new byte [sample length][2]

A mono 16 bit sound can be stored in a word array. A stereo 16 bit sound can be stored in a two dimensional word array. Each element of an 16 bit sound array will be a signed sample value with 0 indicating zero output.

mono16_array = new word [sample length]
stereo16_array = new word [sample length][2]

Instructions are available for loading, saving and playing a sound as an array.

Array operators

A number of operators are available for finding out the characteristics of an array. These may be used in any floating point expression.

*variable***.type** gives the type of the array (**BYTE**, **WORD**, **FLOAT**).
*variable***.size** gives the number of dimensions in the array.
*variable***.dim [** *dim n* **]** gives the number of elements in the specified dimension.

Strings

Formula Graphics can handle strings of characters. Each character in a string is represented by a byte value. The value of each keyboard character was standardised across the computer industry many years ago. The byte value of a character may be used by enclosing it in single quotation marks, for example 'A' has the value 65.

A byte array can be used to store a string of characters. A byte array variable is used as a string variable by preceding its name with a '$'. A string variable can be assigned with a string expression:

$*string variable* **=** *string expression*

A string expression may be a group of characters inside of quotation marks, it may be a string variable, a global string variable, a floating point value, or any combination of these separated by commas. A floating point value will be automatically converted into a string. Strings separated by commas will be combined to form a single string.

When a string variable is assigned with a string expression, a byte array is allocated and assigned to the variable. The size of the array will be equal to the length of the string expression plus one. The string expression will be copied into the array and a zero value placed after it to indicate the end of the string. The following are examples:

$string_one = "This is a string"
$string_two = $string_one, " and the first character value is ", string_one[0]

In the above examples, a byte array called "string_one" will be allocated with 17 elements, then the string will be copied into it. A second array called "string_two" will then be allocated with enough bytes to store the first string plus the rest of the sentence. The letters "84" will be appended to represent the character value of 'T'.

Multidimensional string arrays

If a byte array is allocated with more than one dimension, it can be used to store multiple strings.
The last dimension of the array will be used to store each string and must have enough bytes to store the longest string plus one for the zero value. When a string in a multidimensional array is referred to, no index value needs to be given for the last dimension.

$*string variable* **[** *index 1* **] ... [** *index m-1* **] =** *string expression*

Any number of strings can be copied into a multidimensional array using the same instruction. Only the index values of the first string need to be specified, all other strings will be copied into consecutive positions. Semicolons are used as separators.

$*string variable* **[** *index 1* **] ... =** *string expression**; string expression**; ...*

The following example allocates a two dimensional byte array which can be used to store five strings of up to 9 characters each. Five specified strings will then be copied into the array.

string_array = byte[5][10]
$string_array[0] = "string 1"; "string 2"; "string 3"; "string 4"; "string 5"

String operators and comparisons

A number of operators are available to manipulate strings. The following five operators can be included in any floating point expression. The string parameters used by these operators can be any string expressions.

strlen "example" (returns the length of the string ie. 7)
strval "1.41421" (returns the floating point value of the string, or 0 if not a number)
strcmp $a**;** $b (compares two strings, returns TRUE or FALSE)
stricmp $a**;** $b (compares two case insensitive strings)
strstr $a**;** $b (returns offset of first occurrence of $b in $a, else returns ERROR)

The following three operators can be used in any string expression. The parameters taken by these operators can be any string or floating point expressions.

"example" **stroff** 2 (takes from the given offset in the string ie. "ample")
"example" **strcnt** 4 (takes the given number of bytes from the string ie. "exam")
"example" **stroff** 2 **strcnt** 3 (ie. "amp")

Using strings

A **message** statement can be used to display a string expression. If the program is being run from the Formula Graphics shell, then the string will be displayed in the result window. If it is running with the command line player, the message will be displayed in a Windows message box. Each message will be displayed on a new line.

message *string expression*

The following example shows how the value of a floating point variable could be displayed:

message "The value of x is ", x

CRLF is a predefined string containing the carriage return and line feed control characters. It can be appended to a string to mark the end of a line. For example "first line", CRLF, "second line".

See the **string examples** for more information.

Global strings

Global string variables are similar to string variables, but they do not use byte arrays and their names are preceded by a "%". When a global string variable is assigned with a string, the string will be copied to the program's initialization file in the Windows directory. A global string variable can be permanent.

%*global string variable* **=** *string expression*

A global string variable can be used in a string expression at any time, from any part of a program. An unassigned global string variable will return an empty string.

%account = "Acme"
%region = "North America"
%rate = north_american_rate

The above examples use a global string variables to store configuration information.

Call statements

The best way to write a large program is to break it down into a number of smaller procedures.
A procedure can be defined by specifying a procedure name. All statements following that name with a greater level of indentation are included in the procedure. A **call** statement can be used to jump into a procedure from any place in the program. After the procedure has finished executing, the program will go back to the line following the "call" statement.

A procedure name can be made up of case sensitive letters, numbers and underscores, the first character must be a letter and the name must be followed by a colon. The procedure will begin running at the line following the procedure name. It will run until either the last line has been executed, or a **return** statement is found.

call *procedure*

*procedure***:**
    the procedure starts here
    after it finishes it returns

A procedure can be passed any number of parameters. These parameters can be listed at the end of the "call" statement. If parameters are specified, then the procedure name must be followed by a colon. An equal number of variable names must be listed after the procedure name to take the passed parameters. specified whenParameters may be floating point numbers, arrays, or any other type of data that can be stored using a variable. If the parameter is not a number then it must be preceded by an '@' sign.

call *procedure***:** *parameter 1***,** *@parameter 2***, ...**

*procedure***:** *variable 1***,** *variable 2***, ...**

A procedure can return any number of parameters. These parameters can be listed in a "return" statement. An equal number of variable names must be listed at the beginning of the "call" statement to take the returned parameters. If the parameter is not a number then it must be preceded by an '@' sign.

*variable 1***,** *variable 2***, ... = call** *procedure*

*procedure***:**
    return *parameter 1***,** *@parameter 2***, ...**

Once a variable has been declared, its data will be available throughout the program, including all procedures. That includes variables declared inside procedures and in procedure parameter lists.

For more information, see the **example** on parameter passing.

Scripts

A script is an SXT or other text file containing a list of instruction statements. A script can be executed directly from the menu, or it can be run as part of a presentation. When a script is opened for execution, it is first checked for syntax errors, then it is compiled into a format which can be executed quickly by the computer.

The following instruction can be used to open an instance of a new script from the current script. The new script will be opened, checked and compiled. A handle to the script will then be assigned to the specified script variable.

script variable **= new script** *filename*

Any number of scripts can be opened. Each script will have its own independent variables. One script's variables cannot be touched by, or cannot affect any other script's variables. The variables in each script will remain valid for the life of that script.

Using a script variable, any of the procedures in the specified script can be called from the current script. The calling statement can pass any list of parameters and return any list of parameters.

script variable **call** *procedure***:**
*var 1***,** *var 2***, ... =** *script variable* **call** *procedure***:** *par 1***,** *@par 2***, ...**

Object orientation

More than one script variable can be assigned with an instance of the same script filename. Even though the procedures in each instance of the script will be the same, the variables will still be independent of each other. The technique of opening multiple instances of the same script can be called object oriented programming.

As an example of this technique, consider a script written to describe a moving object. The object's script contains one procedure to initialize the object, one to change its position and one to draw the object at its current position. The position and velocity are stored in variables.

Any number of these objects could be created and maintained by allocating an instance for each one, and storing the instances on a list. To update the position of each object, get its instance from the list and call the object's "change position" procedure and its "draw" procedure.

Opening a second script using the same filename will be faster than the first because the instructions will already be checked and compiled.

Lists

A list can be used to store any number of items. Items can be added, inserted, removed, or retrieved from a list. The following instruction can be used to create a new list and assign it to a variable. Initially, the list will have zero items.

list variable **= new list**

Arrays, scripts, or almost any other type of data that can be assigned to a variable can be placed on a list. Floating point values are the exception, they cannot be placed on a list. Lists can be added to other lists to form trees of data.

list variable **add** *data variable*
data variable **=** *list variable* **get** *position*
list variable **insert** *data variable* **at** *position*
list variable **remove** *position*

These instructions can be used to manage a list. Items are added to the top of the list. When an item is inserted at a position in the list, any items above that position are shifted up one to make a space. When an item is removed, the items above a shifted down. The current number of items on a list can be obtained using the *list variable***.size** operator.

List identification strings

When a list is created, it may also be given an identification string. If that list is then stored in another list, or in a list of lists, then it can be directly retrieved using only its identification string. The following instruction will search through every branch of a list tree, and return the first list with the specified identification string.

list variable **= new list** *identification string*

list variable **=** *base list variable* **get list** *string identifier*

A list identification string can be used in a string expression by preceding the list variable name with a "$". A list without an identification string will return an empty string.

List loops

A list loop can be used to count through every item in a list, getting a handle to each item as it goes. All following statements with a greater level of indentation will be included in the loop.

data variable **=** *list variable* **loop** *count variable* { **type** *type* }{ **mode** *mode* }

The first item on the list will be assigned to the data variable and the count variable will be assigned with zero, then all following statements with greater indentation will be executed.
Then the next item on the list will be assigned to the data variable and the count variable will be incremented by one. The loop will continue until every item on the list has been counted.

An optional variable type may be specified. If a type value is specified, then only variables of that type will be included in the loop. An optional mode value may be specified. The default mode is **ROOT**, for which only the root list in a list tree will be included the loop. The other mode is **TREE** for which every item in every branch of a list tree will be included in the loop.

Data types

When a variable is assigned with any type of data, then that variable becomes a handle to the data. When a variable is added to a list, then the list becomes a second handle to same piece of data. Any number of handles can exist for the same piece of data. A second handle can also be created by direct assignment.

second variable **=** *first variable*

The following statement can be used to copy a piece of data. A separate but identical piece of data will be created and assigned to the new variable. Currently this statement can only be used to duplicate arrays.

copy *new variable* **=** *old variable*

Once a variable has been assigned a floating point value, then it can be reassigned any number of times with another value, but it can only ever be used again for floating point. If a variable is used for any other type of data, then it can only ever be reassigned with data of the same type.

If a variable has been assigned any other type of data than floating point, then it can be unassigned using a **free** statement. After a variable has been freed, it can then be reassigned with any other type of data.

free *variable*

If a variable is freed or reassigned, and if the data that was referred to by the variable has no other handles (if no other variables refer to it and if it is not contained in any list or referred to in other script), then it is no longer in use and it will be destroyed by Formula Graphics.

The *variable***.type** operator can be used to find out what type of data has been assigned to a variable. Its value will be equal to one of the following defined constants:

VARIABLE floating point variable
CONSTANT floating point constant
BYTE_ARRAY byte array
WORD_ARRAY word array
FLOAT_ARRAY floating point array
SCRIPT script variable
LIST list variable
SPLINE spline function
SPRITE sprite variable

Operators, functions and constants

Floating point numbers use 32 bits: 1 for the sign, 8 for the exponent, and 23 for the numerical value. Their range is between +3.4e38 and -3.4e38 with at least 7 digits of precision.

A wide range of arithmetic and logical operators are available for use in floating point expressions. Most of these operators are identical to those available in the C language. They are listed here in their order of precedence:

Parenthesis **()**
Unary minus **-**
Boolean not **!** (returns zero if not zero, or one if zero)
Power operator **^**
Round up with significance **~** (value ~ order of significance)
Round down with significance **_** (value _ order of significance)
Functions
Multiplication and Division ***** **/**
Floating point remainder **%**
Addition and subtraction **+** **-**

Binary shift **<<** **>>** (value >> shift bits)
Binary AND **&**
Binary OR **|**

Equal and not equal **==** **!=**
Magnitude operators **<** **>** **<=** **>=**
Boolean AND and OR **&&** **||**

A wide range of arithmetic, trigonometric and logical functions are also available. Once again, most of these are identical to those available in the C language:

abs positive value
mod gives 1 for +ve, 0 for 0, -1 for -ve
frac fractional component
ceil round up
floor round down
deg degrees to radians
rnd random number in a specified range

sin sine
cos cosine
tan tangent
asin inverse sine
acos inverse cosine
atan inverse tangent
sqrt square root
log log base ten
ln natural log

isdigit returns TRUE for characters '0' to '9'
isalpha returns TRUE for characters 'a' to 'z' or 'A' to 'Z'
tolower converts a character to lower case
toupper converts a character to upper case

These common contants have also been defined and can be used in any floating point expression:

ON 1
OFF 0
TRUE 1
FALSE 0
ERROR -1
PI 3.141592
PI_2 1.570796

The graphics window

The size and color resolution of the graphics window are specified in the graphics options. If the window is opened by the presentation system then the size and color resolution are specified in the presentation options. Information about the graphics window and video mode are stored in the following constants:

GRAPHICS_XMAX *width of the graphics window*
GRAPHICS_YMAX *height of the graphics window*
GRAPHICS_WINDOW *the graphics window handle (Microsoft Windows)*
GRAPHICS_WINDOW_BITS *graphics window color resolution (8, 16 or 24)*
GRAPHICS_DEVICE *the graphics window device context (Microsoft Windows)*
GRAPHICS_DEVICE_BITS *video mode color resolution (bits of color)*

When a bitmap is displayed, its pixels are first copied to the memory buffer of the graphics window. Then, depending on the graphics window update mode, the contents of this buffer will be copied to the video card using the Windows display drivers. The bitmap will not actually appear on the video screen until it is copied to the video card.

If the graphics window update mode is set to **OFF** then the pixels will not be copied to the video card. If the mode is **INVALIDATE** then the pixel area will be invalidated, but will not be copied to the video card. After any number of areas have been invalidated, the combined areas can all be copied to the video card using the **update window** instruction.

If the mode is set to **REFRESH**, then the pixels will always be copied to the video card. This is the default mode.

Sprites

A sprite is an object which has a bitmap, a position, and an optional transparency color. If a sprite is displayed at a position in the graphics window, and if it is later displayed again at another position, the old image will be removed as the new image appears. A sprite can be used to produce a smoothly animated image.

More than one sprite can be animated in the graphics window at one time. If two or more sprites are displayed over the same area of the window they will be displayed in order if their specified depths. The collision of two sprites can also be detected. The following statement will create a new sprite and assign it to the specified sprite variable.

sprite variable **= new sprite**

Before a sprite can be displayed, it must be given a bitmap and a position. If the bitmap is 256 colors and does not use the currently displayed palette then a different palette can be specified. Specifying a palette is not recommended because the remapping process is computationally expensive.

sprite *sprite variable* **bitmap** *array* { **palette** *array* }{ **trans** 0,0,0 } **at** 0,0 (**depth** 0)

An optional transparency color can be given. If the bitmap is 256 colors then the index whose color is nearest to the given red, green and blue components will be transparent. The default depth value is zero, the higher the depth value, the closer the image will be to the top.

Once all the sprites have been declared and assigned bitmaps and positions, they can be displayed using the **update sprites** statement. Every existing sprite will be displayed simultaneously.

Sprites can be tested for collisions. The following statement can be used to test if a sprite has collided with any other existing sprites, the values TRUE or FALSE will be assigned to the result variable. If a second sprite is specified then only those two sprites will be tested against each other.

variable **= collision** *mode* **sprite** *sprite 1* { **sprite** *sprite 2* }

There are two modes for testing collision. The INTERNAL mode can be used to test if the area of one sprite is entirely engulfed by another. The EXTERNAL mode will test whether the two sprites overlap at all.

For more information, see the **example** on playing an animated sprite.

Instruction Index

{ } option
( ) option with default
< > repeatable option
0 any value

Conditional statements

if *condition* { then *variable* = *expression* }
else if *condition* { then *variable* = *expression* }
else { *variable* = *expression* }
switch *value* <, *value* >
case *value* <, *value* >
switch *"string"* <; *"string"* >
case *"string"* <; *"string"* >

Loop instructions

while *condition*
for *variable* <,*variable*> = 0<,0> to 0<,0> { step 0<,0> }
continue
break

Array instructions

array = new byte [i]<[j]>
array = new word [i]<[j]>
array = new float [i]<[j]>
set *array*[i]<[j]> = *value* size 0<,0>
set *array*[i]<[j]> + *value* size 0<,0>
set *array*[i]<[j]> * *value* size 0<,0>
set *array*[i]<[j]> = *array*[i]<[j]> size 0<,0>
set *array*[i]<[j]> + *array*[i]<[j]> size 0<,0>
set *array*[i]<[j]> * *array*[i]<[j]> size 0<,0>
array[i]<[j]> = *value* <, *value* >
$*array* <[j]> = *"string"* <; *"string"* >
copy *array* = *array*
free *array* <,*array*>

Text handling instructions

load *"filename"* byte *array*
save *"filename"* byte *array*
save *"filename"* string *"string"*
variable = parse *array*[i]<[j]> for *"string"*
variable = parse *array*[i]<[j]> to $*array*<[i]>

List handling instructions

list = new list { *"name"* }
list add *variable*
variable = *list* get *pos*
variable = *list* get list *"name"*
variable = *list* loop *count* (type ALL)(mode ROOT)
list insert *variable* at *pos*
list remove *pos*

Scripts and procedures

procedure: { *variable* <, *variable* >}
script = new script *"filename"*
{ *variable* <, *variable* > = } call *procedure* {: *parameter list* }
{ *variable* <, *variable* > = } *script* call *procedure* {: *parameter list* }
return { *parameter list* }
exit

Windows message handling

peekmessage *variable*, *variable*, *variable*, *variable*

Interface instructions

dialog *"filename"* save $*array*<[j]>
dialog *"filename"* load $*array*<[j]>
dialog *"filename"* string $*array*<[j]>
mouse position 0,0
mouse mode 0
message *"string"*
error *"string"*

File management

$*array*<[j]> = get directory
new directory "*dirname*"
copy *"filename"* to *"filename"*
delete *"filename"*
variable = get diskspace *"dirname"*
variable = get filesize *"filename"*

Operating system

MCI *"string"*
DDE *"name"* string *"string"*
set profile string *"filename"*; *"string"*; *"string"* = *"string"*
$*array*<[j]> = get profile string *"filename"*; *"string"*; *"string"*
restart system
execute *"filename"*
delay *value*

Presentation system

display element *"element"*
undisplay element *"element"*
undisplay all
set condition *"string"*
update screen
display screen *"name"*
set textbox *"element"* = *"string"*
set editbox *"element"* = *"string"*
index = listbox *"element"* add *"string"*
state = listbox *"element"* test *index*
set listbox *"element"* (position -1) = *state*
reset listbox *"element"*
set graph *"element"*: *parameter list*
set webdoc *"element"* to *page*
set hypertext *"element"* to *page*
search hypertext *"element"*
getscreen

getrange
testcode
getedit

Bitmap instructions

load *"filename"* bitmap *array* { palette *array* }(frame 0)
save *"filename"* bitmap *array* { palette *array* }(mode 0){ background *array* }
display bitmap *array* { palette *array* }{ trans 0,0,0 }(at 0,0)
subtract bitmap *array* { palette *array* } color 0,0,0 (at 0,0)
capture bitmap *array* { palette *array* }(area 0,0,0,0)(mode 0)
remap bitmap *array* palette *array*
flip 0 bitmap *array*
antidither *value* (area 0,0,0,0)

Palette instructions

display palette *array*
capture palette *array*
compose palette *array*
compose color 0,0,0
analyse (area 0,0,0,0){ bias 0 color 0,0,0 }
optimize 0 (foreground 0)

Sprite instructions

sprite = new sprite
sprite *sprite* bitmap *array* { palette *array* }{ trans 0,0,0 } at 0,0 (depth 0)
variable = collision 0 sprite *sprite* { sprite *sprite* }
update sprites

Graphics window

print window
update mode 0
invalidate area 0,0,0,0
update window
set window (area 0,0,0,0)(color 0,0,0)
MCGA mode 0
set pixel 0,0 { to 0,0 } color 0,0,0
polygon 0,0 <;0,0> color 0,0,0
polygon 0,0 ;0,0 ;0,0 ;0,0 bitmap *array* { palette *array* }
variable, *variable*, *variable* = get pixel color 0,0
variable, *variable* = get pixel position 0,0,0

Bounding instructions

{ *variable* = } bound *variable* <, *variable* > to 0,0<;0,0> (mode CLIP)
bitmap *array* at *variable*, *variable* area 0,0,0,0 at 0,0 (mode TEST)

Sound instructions

load *"filename"* sound *array* rate *variable*
save *"filename"* sound *array* rate 0
play sound *array* rate 0
wait sound
stop sound

Instruction Reference

Condition statements

if *condition* { **then** *variable* **=** *expression* }

If the value of *condition* is not equal to zero then all following statements with greater indentation will be executed. If the value is equal to zero then those statements with greater indentation will be skipped. An optional variable assignment may be specified instead of using indented statements.

else if *condition* { **then** *variable* **=** *expression* }

If the result of the previous condition statement with the same indentation was false, then an alternative condition can be specified.

else { *variable* **=** *expression* }

If the result of the previous condition statement with the same indentation was false, then alternative statements with greater indentation or an optional variable assignment can be executed.

switch *value* <**,** *value* >
case *value* <**,** *value* >

A "switch" instruction can be used to specify a one or more floating point values. Following this statement will be a number of indented "case" statements, each "case" statement must specify the same number of floating point values. When the "switch" statement is executed, each "case" statement will be tested to find one with equal values. If such a case is found then all statements with greater indentation following this will be executed. If no such case is found and if a **default** case has been declared then all indented statements following this will be executed. Every other "case" statement will be skipped.

```
switch val
        case 0
                execute these statements
        case 1
                execute these statements
        ...
        default
                execute these statements
```

A "switch" statement can also be used to specify one or more strings. Each case statement will be tested to find one with case insensitive matching strings. If matching strings are found then all statements following this case with greater indentation will be executed.

switch *"string"* <**;** *"string"* >
case *"string"* <**;** *"string"* >

Loop instructions

A "while" loop will continue to execute all following statements with greater indentation for as long as the value of the specified condition is not equal to zero.

while *condition*

A "for" loop will increment the value of each loop variable from its starting value to its finishing value. If more than one variable is specified then the loops will be nested with the first variable in the outer loop. All following statements with greater indentation will be executed by the loop.

for *variable*<,*variable*> **=** 0<,0> **to** 0<,0> { **step** 0<,0> }

If a "continue" statement is found during the execution of a loop, the current iteration of the loop will finish, and the next iteration will begin at the top of the loop.

continue

If a "break" statement is found during the execution of a loop, the loop will immediately finish, and execution will jump to the first line after the loop.

break

Array instructions

Each of the following instructions will allocate an array, the elements of the array will be set to zero, and the array will be assigned to the variable: The array can be have any number of dimensions. Each dimension can have any number of elements.

array **= new byte** [i]<[j]>
array **= new word** [i]<[j]>
array **= new float** [i]<[j]>

A section of any type of array can be set to a specified value. The section will begin at the given element and extend as far as the given sizes in each dimension. Each element in the section can also be added or multiplied by a specified value.

set *array*[i]<[j]> **=** *value* **size** 0<,0>
set *array*[i]<[j]> **+** *value* **size** 0<,0>
set *array*[i]<[j]> ***** *value* **size** 0<,0>

The following example will set the 3rd, 4th and 5th indexes of a byte array to space characters.

set my_string[3] = ' ' size 3

A section of any type of array can be copied to a section of the same or any other type of array. The sections will begin at the given elements and extend as far as the given sizes in each dimension. The elements in the first section may also have the elements in the second section added or multiplied to them.

set *array*[i]<[j]> **=** *array*[i]<[j]> **size** 0<,0>
set *array*[i]<[j]> **+** *array*[i]<[j]> **size** 0<,0>
set *array*[i]<[j]> ***** *array*[i]<[j]> **size** 0<,0>

Using the "copy" instruction another handle can be assigned to the same array.

copy *array* **=** *array*

Using the "free" instruction, if there are no other existing handles to the specified array, then the array will be deallocated. In any case the handle will be freed. More than one array can be freed by the same instruction.

free *array* <,*array*>

Text handling instructions

The contents of any file can be loaded as a one dimensional byte array. The array will be allocated with enough space for the entire file. The contents of the file will be loaded into the array and the array will be assigned to the specified variable.

load *"filename"* **byte** *array*

The entire contents of a byte array can be saved to a file. The file name can be any string expression.

save *"filename"* **byte** *array*

A string can be added to the end of a text file. If the file does not exist it will be created. The string can be any string expression.

save *"filename"* **string** *"string"*

A byte array can be searched for a given string. The search will begin at the specified array index and will continue until either the string is found, or the end of the array is reached. If the string is found, the given variable will be assigned with the first index of the search string in the array. If the string is not found, the variable will be assigned with the value ERROR.

variable **= parse** *array*[i]<[j]> **for** *"string"*

The text in a byte array can be broken down into terms. A term may be a floating point number, a symbol (case sensitive letters, numbers and underscores, the first character must be a letter) or a quote (any characters contained in quotation marks). Spaces and punctuation marks will be ignored.

variable **= parse** *array*[i]<[j]> **to** $*term*<[i]>

This instruction will begin searching at the specified index of the array. If a term is found, it will be copied into the given destination string and the given variable will be assigned with the first index after the term. When there are no more terms to be found, the variable will be assigned with the value ERROR.

List handling instructions

The following instruction can be used to create a new list and assign it to a variable. An optional identification string can be specified for the list.

list **= new list** { *identifier* }

The data referred to by a variable can be added to the list.

list **add** *variable*

A variable can be assigned with data from a given position in the list.

variable **=** *list* **get** *pos*

All the branches of a parent list can be searched for the first occurrance of a child list with the given identification string.

variable **=** *list* **get list** *identifier*

A list loop can be used to retrieve a handle to every item on a list. The loop will begin each iteration by assigning an item to the given variable, then all following statements with greater indentation will be executed. After each iteration, the counter variable will be incremented. The loop has two modes, the ROOT mode is the default and will only include the items in the root of the given list. The TREE mode will also include all items in all child lists of the given parent list.

variable **=** *list* **loop** *counter* (**type** ALL)(**mode** ROOT)

When an item is inserted at a position in the list, any items above that position are shifted up one.

list **insert** *variable* **at** *pos*

When an item is removed from a position in the list, any items below that position are shifted down one.

list **remove** *pos*

Scripts and procedures

A new procedure can be defined using a procedure name. All indented statements after the name will be included in the procedure. Any number of variables can be listed after the name to take data that is passed to the procedure.

*procedure***:** { *variable* <**,** *variable* >}

A procedure can be called. If one or more pieces of data are passed to the procedure then they must be separated from the procedure name using a colon. Any type of data that can be stored in a variable can be passed to a procedure. If the data is not floating point, then the variable name must be preceded by an '@' sign.

call *procedure* {**:** *parameter list* }

A "return" statement may be used to return from any place in a procedure. If no "return" statement is found then the procedure will return after the last line anyway. One or more pieces of data can be returned from a procedure by specifying them after the "return".

return { *parameter list* }

If one or more pieces of data are returned from the procedure then a list of variables must be given to take the data.

{ *variable* <**,** *variable* > **=** } **call** *procedure* {**:** *parameter list* }

Using the following instruction, a new script can be opened from the current script. The given variable will be assigned with the new instance of the script. More than one instance of the same script can be opened. Each instance will have its own variable space. The file name can be specified as any string expression.

script **= new** script *"filename"*

A procedure in another script can be called by specifiying the script variable.

{ *variable* <**,** *variable* > **=** } *script* **call** *procedure* {**:** *parameter list* }

A program can be ended at any place by using an exit instruction.

exit

Windows message handling

The following instruction can be used to get the next message from the Microsoft Windows message queue. The message type will be assigned to the *message* variable. The word parameter, and the low word and high word of the long parameter will also be assigned to variables. If no message is available, the message variable will be assigned with zero. This instruction does not wait for a message.

peekmessage *message*, *wparam*, *loword*, *hiword*

The following message types have been defined as constants:

WM_KEYUP any key has been unpressed
WM_KEYDOWN any key has been pressed
WM_MOUSEMOVE the mouse has moved
WM_LBUTTONUP the left mouse button has been unclicked
WM_RBUTTONUP the right mouse button has been unclicked
WM_LBUTTONDOWN the left mouse button has been clicked
WM_RBUTTONDOWN the right mouse button has been clicked
WM_LBUTTONDBLCLK the left mouse button has been double clicked
WM_RBUTTONDBLCLK the right mouse button has been double clicked

If the message is WM_KEYUP or WM_KEYDOWN then the wparam can be used to determine which key was pressed. Alphanumeric keys will return their standard character values. The following keys have also been defined:

VK_SPACE
VK_RETURN
VK_ESCAPE
VK_CONTROL
VK_LEFT
VK_UP
VK_RIGHT
VK_DOWN
VK_PRINT
VK_PAUSE

If the message is WM_MOUSEMOVE or any other mouse message then the position of the mouse cursor can be found in the *loword* and *hiword* variables. The following example illustrates how to detect messages. For more information on Windows messages, see the Microsoft Windows 3.1 SDK documentation.

```
peekmessage m, w, l, h
if m == WM_KEYDOWN
     if   w == 'A'
          then the letter 'A'   was pressed
if m == WM_KEYDOWN
     if   w == VK_PAUSE
          then the pause key was pressed
if m == WM_LBUTTONDOWN
     then the left mouse button was pressed
if m == WM_MOUSEMOVE
     x_mouse_position = l
     y_mouse_position = h
```

Interface instructions

A load file dialog box can be opened. A dialog box with the specified title string will allow the user to browse the directories and choose a file name and path. The given byte array may be used to give a default file name and path. The user selected file name and path will be copied back into the byte array. If the cancel button is pressed, the array will be returned with an empty string.

dialog *"title"* **load** $*array*<[i]>

A save file dialog box can be opened. A dialog box with the specified title string will allow the user to browse the directories and choose a file name and path. The given byte array may be used to give a default file name and path. The user selected file name and path will be copied back into the byte array. If the file name already exists the user will be prompted. If the cancel button is pressed, the array will be returned with an empty string.

dialog *"title"* **save** $*array*<[j]>

An input dialog can be opened with a single line edit control. The string contents of the specified byte array will be displayed in the edit control of a dialog box with the specified title string. The string can be modified and returned to the byte array.

dialog *"title"* **string** $*array*<[j]>

The mouse cursor can be set to any position in the graphics window. The new horizontal and vertical pixel position of the cursor hotspot is specified.

mouse position 0**,**0

The mouse cursor can be disabled. If the mode value is OFF then the mouse cursor will disappear from the video display. If the mode value is ON then the mouse cursor will reappear.

mouse mode *mode*

A string expression can be displayed in the result window. If the program is being run from the Formula Graphics shell, then the string will be displayed in the result window. If it is running with the command line player, the message will be displayed in a Windows message box. Each message will be displayed on a new line.

message *"string"*

An error report can be used to stop the execution of the program. The error string expression will either be sent to the result window or displayed in a Windows message box.

error *"string"*

File management

The current operating directory can be found and copied into a specified string.

$*array*<[j]> **= get directory**

A new directory can be created. The following instruction will create a new directory with the given name.

new directory *"name"*

A file can be copied from a source to a destination. Both the source and destination strings can contain a full path. If the file was compressed using the Microsoft "compress" utility it will decompressed as it is copied.

copy *"filename"* **to** *"filename"*

The following instruction will delete the specified file. The filename may be given as any string expression.

delete *"filename"*

A disk drive can be checked for the amount of free space. Only the first character in the directory name string will be used to determine which disk drive to look at.

variable **= get diskspace** *"directory"*

The size of any file can be determined. The size will be assigned to the given variable.

variable **= get filesize** *"filename"*

Operating system

Microsoft Windows MCI (Multimedia Control Interface) can be used to play sound and midi files as well as control VCR machines and show animations. The following instruction can be used to execute an MCI command.

MCI *"string"*

The following example uses the MCI to play an AVI animation with sound. For more details on MCI see Microsoft multimedia documentation.

```
MCI "open EXAMPLE.AVI alias example parent ",HWND," style child"
MCI "window example state show"
MCI "play example wait"
MCI "close example"
```

Windows applications can communicate with one another using DDE strings. The following example shows how the DDE instruction can be used to create a new program group in Program Manager and install an icon. For more details on installing icons, see Microsoft Windows 3.1 SDK documentation.

DDE *"name"* **string** *"string"*

```
DDE "PROGMAN" string "[CreateGroup(",$title,")]" // Create the program group
DDE "PROGMAN" string "[ShowGroup(1)]" // Display the new group
DDE "PROGMAN" string "[AddItem(",$command,",",$title,",",$icon,")]" // Create the icon
```

The options in any Windows initialization file can be set or changed. The specified filename must be an existing INI file in the Windows directory. This instruction may also be used on WIN.INI and SYSTEM.INI.

set profile string *"filename"***;** *topic***;** *option* **=** *"string"*

Any option in any Windows initialization file can be found and copied into a specified string.

$*array*<[j]> **= get profile string** *"filename"***;** *"string"***;** *"string"*

The following instruction will restart Windows. If any changes are made to WIN.INI or SYSTEM.INI then Windows must be restarted before the changes can take effect.

restart system

Another application can be loaded into memory and executed by the following command. Only one copy of Formula Graphics can be running in Windows at any one time.

execute *"filename"*

A time delay can be specified in hundredths of a second. The delay will be accurate to plus or minus one hundredth of a second.

delay *value*

The **TIME** variable can be used to determine the number of hundredths of a second that have elapsed since the program began executing.

Presentation system

The following command can be used to play or display an element in the current screen. If the element has already been displayed then it will be redisplayed.

display element *"element"*

An element can also be undisplayed, or every element on the screen can be undisplayed. Background elements will not be removed.

undisplay element *"element"*
undisplay all

The expression **getedit** "element" can be used in a string expression to get the contents of the edit box with the given name. The following instruction will copy the specified string into the edit box element with the given name.

set editbox *"element"* **=** *"string"*

The text inside in a text element can be changed. The old string will be undisplayed and the new string will be displayed.

set textbox *"element"* **=** *"string"*

A string can be added to a list box. The specified string will be added to the list box with the given name and the position of the string in the list will be assigned to the *index* variable.

index **= listbox** *"element"* **add** *"string"*

The following instruction can be used to test the selection state of any string in a list box. If the string at the given position is selected then the given variable will be assigned ON, if not it will be assigned OFF.

state **= listbox** *"element"* **position** *index*

The following instruction can be used to set the selection state of any string in a list box. Each string can either be set to ON or OFF. If no position value is given, or if the position value is -1, then the selection state of every string will be set.

set listbox *"element"* (**position** -1) = *state*

The following instruction will reset the contents of the list box with the given name. The list box will become empty.

reset listbox *"element"*

The following instruction can be used to display a specified page in a hypertext document. Any condition codes associated with all other pages will be switched off and any code associated with the specified page will be switched on.

set hypertext *"element"* **to** *page*

The following instruction will open a Text Search dialog box for the given hypertext element. Every keyword in the document will be listed. If one of these keywords is selected, then the page associated with it will be displayed. If more than one page specifies the same keyword, then another dialog box will open and the page numbers or the topics associated with those pages will be listed.

search hypertext *element*

If any word phrase is typed into the Text Search dialog box and the Search button is pressed, then the document will be scanned for any case insensitive occurrences of that phrase. If more than one page has an occurrence of the phrase, then another dialog box will open and the page numbers or topics associated with those pages will be listed.

The following instruction can be used to update a graph element. Five paramaters must be passed to the graph. The first parameter must be an array containing the data values. The second parameter will be an array containing the graph's colors. The third parameter will be the minimum value, the fourth parameter will be the maximum value and the fifth value will be the increment.

set graph *"element"***:** *parameter list*

The expression **testcode** *code* can be used in any floating point expression to return the state of the given condition code. The result will be equal to TRUE or FALSE.

The expression **getrange** *"range"* can be used in any string expression to return a list of all TRUE condition codes in the given range. The result will be a string which lists the condition codes separated by commas. If the range string is empty then all true condition codes will be listed.

The following instruction can be used to set a condition code, a number of condition codes, or a range of condition codes.

set condition *"string"*

After the condition codes have been changed, the screen may need to be updated to reflect their new state. Elements which no longer have valid test conditions will be undisplayed and elements with newly valid codes will be displayed.

update screen

The following instruction will return to the presentation system. The presentation system will then jump straight to the specified screen.

display screen *"name"*

The expression **getscreen** can be used in any string expression to return the name of the current screen.

Bitmap instructions

A bitmap file of any format can be loaded into an array. An array will be allocated, the bitmap file will be decoded and copied into the array, and the array will be assigned to the given variable. Depending on the color resolution of the bitmap, the array will be a two or three dimensional byte or word array. An optional palette array can be specified for a bitmap with a palette. If the specified file is an animation file, then a frame number can be also be given.

load *"filename"* **bitmap** *array* { **palette** *array* }(**frame** 0)

A bitmap array can be saved as a bitmap file of any format. The color resolution of the bitmap array must be compatible with the given file format. If the bitmap is 256 colors then a palette array must also be specified. If an animation file is specified, then the array will be added as a frame to the end of the animation file. The four available animation modes are SIMPLE, DELTA, OVERLAY or SPRITE. The last two modes require a background bitmap array to be given.

save *"filename"* **bitmap** *array* { **palette** *array* }(**mode** SIMPLE){ **background** *array* }

A bitmap array can be displayed on the graphics window. If the color resolution of the bitmap array is different from the color resolution of the graphics window, then the array will be converted before it is displayed. If the bitmap is 256 colors and if its palette is different from the currently displayed palette, then its palette array may be specified. If a palette is specified, then the color indexes of the bitmap array will be remapped so that the colors in the specified palette array match the colors in the current palette as closely as possible.

The red, green and blue components of a transparency color may also be speciifed. Any pixels in the array with this color will not be displayed. If the bitmap array is 256 colors, then a transparent index will be chosen that matches the red green and blue values of the transparency color as closely as possible.

display bitmap *array* { **palette** *array* }{ **trans** 0,0,0 }(**at** 0,0)

The image in the graphics window can have a bitmap array subtracted from it. All pixels in the graphics window which are the same color as the pixels in the bitmap array will be set with the specified color. Only pixels which are different in color will be unchanged.

subtract bitmap *array* { **palette** *array* } **color** 0,0,0 (**at** 0,0)

The contents of the graphics window can be captured into a bitmap array. An array will be allocated large enough to store a bitmap of the specified area, then the array will be assigned to the given variable. The area to capture can be specified as the X and Y position and the X and Y lengths. If no area is specified then the whole graphics window will be captured. The currently displayed palette can also be captured and assigned to the given palette array variable.

There are four modes of capture. If the mode is COPY then the captured bitmap will have the same color resolution as the graphics window. If the mode is PAL then the captured bitmap will be converted to a two dimensional byte array in a 256 color format. If the mode is RGB16 then the bitmap will be converted to a two dimensional word array in 16 bit color. If the mode is RGB24 then the bitmap will be converted to a three dimensional byte array in 24 bit color.

capture bitmap *array* { **palette** *array* }(**area** 0,0,0,0)(**mode** COPY)

A 256 color bitmap array can be remapped to the currently displayed palette. The closest matching colors will be found for the colors in the given palette array, then the color indexes in the bitmap array will be changed. After remapping, the bitmap array can be displayed directly without having to specify its palette.

remap bitmap *array* **palette** *array*

A bitmap can be flipped in either the HORIZONTAL or VERTICAL mode.

flip *mode* **bitmap** *array*

An area of the graphics window can be processed with an antidither filter. If no area is specified then the entire window will be antidithered.

antidither *value* (**area** 0,0,0,0)

For more details on using arrays as bitmaps, read the discussion on arrays.

Palette instructions

A new palette can be displayed to the graphics window. In most cases all 256 colors in the specified palette will be copied to the current graphics window palette.

If the graphics window is 256 colors and the video card is using a 256 color video mode, then only the first 236 colors in the palette array will be copied to the current system palette. These colors will be copied to palette positions 10 to 245 leaving the 20 Windows system colors unchanged. If the palette array already contains the Windows system colors then only the 236 colors starting at position 10 in the palette array will need to be copied to the current system palette.

display palette *array*

The current graphics window palette can be captured and stored in a two dimensional byte array. In all cases, all 256 colors will be copied into the array and the array will be assigned to the given variable.

capture palette *array*

The individual colors in a palette can be allocated by the palette management system. If there are enough vacant positions in the system palette then each unique color in the specified palette will be allocated a palette position. When the script that allocated the palette is freed then its colors will be deallocated.

compose palette *array*

A single color can be allocated by the palette management system. That color will be allocate a vacant position in the system palette. When the script that allocated the color is freed then the color will be deallocated.

compose color 0,0,0

An area of the graphics window must be analysed before an optimum palette is found. If no area is specified then the entire screen will be analysed. Bias can be given towards a particular color group. The analysis data will accumulate each time this instruction is used. This data will be cleared when an "optimize" instruction is executed.

analyse (**area** 0,0,0,0){ **bias** 0 **color** 0,0,0 }

The "optimize" instruction will use any existing analysis data to find the optimum palette. By default, the new palette will be made up of the most popular colors. A optional number of foreground colors can be made a subset of the new palette. Foreground colors are a wide spectrum of colors to ensure that all of the different color groups are represented.

optimize *colors* (**foreground** 0)

Sprite instructions

A new sprite can be created with the following statement.

sprite **= new sprite**

A sprite variable can be given a bitmap, a palette, a transparency color, a position and a depth value. The sprite will not be displayed by this statement.

sprite *sprite* **bitmap** *array* { **palette** *array* }{ **trans** 0**,**0**,**0 } **at** 0**,**0 (**depth** 0)

If only one sprite variable is specified by the following statement then that sprite will be checked with every other sprite on the screen to determine whether it is in a state of collision. If two sprite variables are specified then only these two will be tested. If there is a collision then the given variable will be assigned the value TRUE, otherwise it will be assigned the value FALSE. There are two modes of collision. The INTERNAL mode can be used to test if the area of one sprite is entirely engulfed by another. The EXTERNAL mode will test whether the two sprites overlap at all.

variable **= collision** *mode* **sprite** *sprite* { **sprite** *sprite* }

The following statement must be used to display the sprite variables. Every sprite variable will be displayed, regardless of which script it belongs to. After new sprites have been created, old sprites have been deleted, sprites have moved or their bitmaps have changed, this instruction can be called to display the new information in the graphics window.

update sprites

Graphics window

The image in the graphics window can be output to the printer.

print window

The graphics window update mode can be set to OFF, INVALIDATE or REFRESH. The default mode is REFRESH. For more details on the modes read the discussion of the graphics window.

update mode *mode*

An area of the graphics window can be invalidated. If the update mode is OFF then this instruction will be ignored. If the update mode is REFRESH then the invalid area will be immediately updated to the video display.

invalidate area *xpos,ypos,xlen,ylen*

All invalid areas will be updated to the video display by the following instruction.

update window

The entire graphics window can be set to any color. An area may also be specified, all pixels in this area will be set to the specified color. If no color is specified, the window will be set to black.

set window (**area** 0,0,0,0)(**color** 0,0,0)

The video mode can be changed to 320 x 200 in 256 colors. The Microsoft multimedia extensions file "dispdib.dll" can be used to switch the video card into MCGA mode 13. All instructions will work the same but the mouse cursor will not be visible. The mode value must be either ON or OFF.

MCGA mode 0

Any pixel in the graphics window can be set to any color. If the position of a second pixel is specified then a line will be drawn between the to pixels. The red, green and blue components of the color must be specified.

set pixel 0,0 { **to** 0,0 } **color** 0,0,0

A polygon can be drawn in the graphics window. Any number of pixel positions can be given as vertices. The red, green and blue components of the color must be specified.

polygon 0,0 <;0,0> **color** 0,0,0

A bitmap array can be displayed using any four pixel positions as its corners.

polygon 0,0 **;**0,0 **;**0,0 **;**0,0 **bitmap** *array* { **palette** *array* }

The color of any pixel in the graphics window can be found. The red, green and blue components of the color will be assigned to the given variables.

*variable***,** *variable***,** *variable* **= get pixel color** 0,0

Information about the graphics window and video mode can be found using the following constants:

GRAPHICS_XMAX *width of the graphics window*

GRAPHICS_YMAX *height of the graphics window*
GRAPHICS_WINDOW_BITS *graphics window color resolution (8, 16 or 24)*
GRAPHICS_DEVICE_BITS *video mode color resolution (bits of color)*

Bounding instructions

The value of a variable can be limited to set boundaries. One or more variables can be specified. Each variable will be compared against its lower and upper limits. If the mode value is TEST then the result variable will be TRUE if the variable is within its limits, and FALSE if it is not. If the mode is CLIP then the values of the variables will not be allowed to pass the limits. If the mode is TILE and the value of the variable passes the upper limit it will be set to the value of the lower limit and visa versa.

{ *variable* **=** } **bound** *variable* <**,** *variable* > **to** *lower*,*upper* <**;** *lower*,*upper* > (**mode** CLIP)

Any working area can be mapped to a bitmap. The specified coordinate in the specified area will be mapped proportionally to the area of the bitmap and the corresponding pixel position will be assigned to the given variables. If the mode is TEST then all values will be proportional. If the mode is CLIP then the pixel position will be limited to the size of the bitmap. If the mode is TILE then the bitmap will be tiled across the area and if it is FLIP then every second tile will be flipped either horizontaly or vertically.

bitmap *array* **at** *variable***,** *variable* **area** 0**,**0**,**0**,**0 **at** 0**,**0 (**mode** TEST)

Sound instructions

A sound file of any format can be loaded into an array. An array will be allocated, the sound file will be decoded and copied into the array, and the array will be assigned to the given variable. Depending on the number of channels and the bits per sample, the array will be a one or two dimensional byte or word array. The sample rate of the sound file will also be assigned to the given variable.

load *"filename"* **sound** *array* **rate** *variable*

A sound array can be saved as a sound file of any format. The sample rate of the sound must also be specified.

save *"filename"* **sound** *array* **rate** 0

A sound array can be played by a sound device. If no sound device exists, the instruction will be ignored. If any other sound is currently playing, it will be immediately stopped.

If the SOUND_DEVICE constant is TRUE, a sound device is present in the system, if it is FALSE then there is no sound device.

play sound *array* **rate** 0

If a sound is currently playing, the SOUND_SAMPLE variable can be used to find the current sound sample number. If no sound is playing, or the sound is finished then the value of this variable will be ERROR.

The following command will halt the execution of the program until the currently playing sound is finished.

wait sound

The currently playing sound can be stopped.

stop sound

If the MIDI_DEVICE constant is TRUE, a midi device is present in the system, if it is FALSE then there is no midi device. Midi devices can be operated using the MCI interface.

Language examples

String manipulation

It is not necessary to allocate the byte array which will be used to store your string. You can simply allocate the string to any old variable name.

```
$my_string = "This is my string"
```

The variable name can then be used as a string by prefixing it with '$' sign:

```
message $my_string                  // displays the string in the result window
$my_new_string = $my_string     // one string can be assigned to another
```

The variable can be used as a string, but deep down inside it is a byte array, and when no prefix is given, it can be used as one:

```
my_string[0] = '?'                      // '?' is used to represent the ISO character value of the question mark
my_string[1] = 0            // a zero value is used to indicate the end of a string.
```

Strings can be appended together by separating two or more strings with commas:

```
$my_new_string = $my_string, " and this is some more"
```

If a number is used where a string was expected, then the number will be automatically converted into a string:

```
$my_string = 10                                 // the value 10 is interpreted here as "10"
x = 10
$my_new_string = $my_string, " = ", x               // the new string would be "10 = 10"
```

The length of a string can be found using "strlen", and in this case the length is displayed as a string:

```
message "The length of my string is ", strlen $my_string
```

Parameter passing

In this example, just about every known type of variable will be passed to a procedure, and just about every known type of variable will be passed back.

```
a = 10                          // Initialize a floating point variable
$b = "my string"                // Initialize a string variable
c = new float[10]               // Initialize a floating point array
c[0] = 0,1,2,3,4,5,6,7,8,9      // Assign some values to the array
d = new list                    // Initialize a new variable list
d add a                         // Throw a couple of variables on the list
d add b

m, n, p, r, s = call my_procedure: @d, @c, @b, a, 10

// m will be a variable list
// n will be a floating point array
// p will be a byte array, and by prefixing it with a '$' it will be a string
// r will be a variable containing the value 10
// s will be a variable containing the value 10
return

my_procedure: e, f, g, h, k
    // e will be a variable list
    // f will be a floating point array
    // g will be a byte array, and by prefixing it with a '$' it will be a string
    // h will be a variable containing the value 10
    // k will be a variable containing the value 10
    return @e, @f, @g, h, k
```

Converting an animation into bitmaps

This example assumes that each frame of the AVI uses 24 bits of color. Each frame is analysed to build a color histogram with a bias towards white. Then the optimum 256 color palette is calculated. Then as each 256 color bitmap is captured from the 24 bit graphics window, its colors are mapped to its palette.

```
for n = 0 to frame_count
    load "animation.avi" bitmap bmp24 frame n          // load a 24 bit frame
    display bitmap bmp24                                    // display the frame
    analyse bias 10 color 255,255,255                       // calculate color histogram
    optimize 256          foreground 64                             // get best 256 colors
    capture bitmap bmp8 mode PAL                       // convert to 256 colors
    capture palette pal8                               // get the new palette
    save "bitmap",n,".gif" bitmap bmp8 palette pal8          // save as a GIF
```

Converting bitmaps into an animation

This example assumes that each frame of the AVI uses 24 bits of color. Firstly, all the BMP's are analysed to build a color histogram with a bias towards white. Then the optimum 256 color palette is calculated. After the first frame has been saved with the new palette, then every other frame can be saved onto the end. Use a 24 bit graphics window.

```
for n = 0 to frame_count step 5                    // not every frame needs to be analysed
load "bitmap",n,".bmp" bitmap bmp24 frame n              // load a bitmap
    display bitmap bmp24                                 // display the bitmap
    analyse bias 10 color 255,255,255                    // build color histogram

optimize 256 foreground 64                  // optimum palette for entire animation

for n = 0 to frame_count
load "bitmap",n,".bmp" bitmap bmp24 frame n         // load a bitmap
    display bitmap bmp24                                 // display the bitmap
    capture bitmap bmp8 mode PAL                         // convert to 256 colors
    capture palette pal8                                 // get the new palette
    save "animation.flc" bitmap bmp8 palette pal8                // save to an FLC
```

Installing a desktop icon

The Windows 3.1 program manager uses the DDE (dynamic data exchange) protocol to communicate with setup applications and to install program icons. This technique works with Windows 3.x, 95 and NT.

```
$target = "formula"
$title = "Formula Graphics"
$destdir = get directory
DDE "PROGMAN" string "[DeleteGroup(",$title,")]"
DDE "PROGMAN" string "[CreateGroup(",$title,")]"
DDE "PROGMAN" string "[ShowGroup(1)]"
DDE "PROGMAN" string "[AddItem(",$destdir,"\",$target,".exe,",$title,",",$destdir,"\",$target,".exe,0)]"
```

Returning graph data

```
graph_element_data:
    data = new float[3][4]                  //allocate data array
    colors = new byte[3][3]                 //allocate color array
    colors[0][0] = 200,130,100              //set colors
    colors[1][0] = 100,180,130
    colors[2][0] = 100,160,210
    for n,m = 0,0 to 2,3
        data[n][m] = 10 + rnd 90            //fill with random data
    return @data, @colors, 0, 100, 10       //return info to graph element
```

Playing an animated sprite

This example assumes that no palette remapping is required. For a more detailed example of animated sprites, see the KillMunger scripts included in the demonstation presentation.

```
files = new byte[4][32]
$files[0] = "image1.bmp"; "image2.bmp "; "image3.bmp"; " image4.bmp"

bitmap_list = new list                          //make a list of bitmaps
for n = 0 to 3
    load $files[n] bitmap bmp                   //load each bitmap
    bitmap_list add bmp                         //add each bitmap to the list
for n = 0 to 10000
    call draw_sprite: xpos, ypos                //move the sprite across the screen
    xpos,ypos = xpos+1,ypos+1
    if xpos > GRAPHICS_XMAX then xpos = 0       //keep the sprite in the window
    if ypos > GRAPHICS_YMAX then ypos = 0
return

draw_sprite: x, y
    bmp = sprite_bitmaps get frame_count        //cycle the bitmaps used for the sprite
    sprite anim_sprite bitmap bmp at x, y depth y       //prepare the sprite
    frame_count = frame_count + 1
    if frame_count > 3 then frame_count = 0
    update sprites                              //display the sprite
```

Programming an Edit box

```
set_edit_box:
    load "information.txt" byte information                 //load a text file
    set textbox "element_name" = information    //copy the text to the edit box

get_edit box:
    log_data = getedit "element_name"           //get the text from the edit box
    save "logfile.txt" string log_data          //append the text to a log file
```

Programming a List box

```
set_list_box:
    items = new byte[5][20]
    $items[0] = "item 1"; "item 2"; "item 3"; "item 4"; "item 5"
    for n = 0 to 4
        listbox "element_name" add $item[n]      //add each item to the list

detect_list box:
    for n = 0 to 4
        state = listbox "element_name" test n    //detect the selection state of each item
        if state == TRUE
            ...
```

Getting data from a database

The database should be exported from its original application as a text file. The text file can then be searched for matching strings. First find the appropriate table, then find the appropriate field within the table. Then each item can be extracted and stored in an array for easy reference.

```
init_database:
    buffer = new byte[32]
    data = new float[100]
    load "database.txt" byte database          // load database as text file
    pos = parse database[0] for $state         // search for the desired table
    pos = parse database[pos] for $account     // search for the desired field
    pos = parse database[pos] to $buffer                  // move to next item
    for n = 0 to 99
        pos = parse database[pos] to $buffer   // store data in an array
        data[n] = strval $buffer
```