Mildred

COLLABORATORS			
	TITLE : Mildred		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 8, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Mildred

1.1 Mildred v1.51

```
                Here is the documentation for Mildred v1.51, released 3rd March  ↩
                   2000.

Mildred v1.51 and earlier versions are Copyright (c) 1998-2000 Paul West.
Written by Paul West.


                 Information
                 <<-- READ THIS FIRST, IMPORTANT!

                 History
                 cpu


                 MCPU
                 Processor.b" ; Set cpu routines allowed to use. 0..3=000..030+,  ↩
                    or >=4=040+. CAREFUL!!

                 Mc2pCPUmode
                 CPU.b ; Set cpu c2p uses. Use 'Processor' or MProcessor. <4=030-, ↩
                     >3=040+

                 MProcessor
                 ; Returns value 0..6 representing MC68000..MC68060 cpu according  ↩
                    to exec\AttnFlags

 c2p


                 MReservec2pWindows
                 [()]NumberOfWindows.w[)] ; Reserve structure-memory for c2pWindows

                 MAutoUsec2pWindows
                 True/False ; Automatically 'use' new c2pWindows. <>0=True

                 Mc2pWindow
                 c2pWindow#.w,OpWidth.w,OpHeight.w[,SourceBWidth.w[,Processor.b], ↩
                    PlanarWidth.w,PlanarHeight.w]
```

```
                Mc2pWindowWidth
                [(c2pWindowNumber.w)] ; Returns width of c2pWindow

                Mc2pWindowHeight
                [(c2pWindowNumber.w)] ; Returns height of c2pWindow

                Mc2pWindowNewHeight
                c2pWindow#.w,NewHeight.w ; Change height of already defined c2p  ↩
                    object

                Mc2p
                [[c2pWindow#.w],Chunky.l],Planar.l ; Convert chunky to planar ( ↩
                    Use Mc2pWindow first)

                MFreec2pWindow
                [Firstc2pWindow.w[,Lastc2pWindow.w]] ; Free a c2pWindow, range of ↩
                     c2pWindows or all c2pWindows

                MUsec2pWindow
                Mainc2pWindowNum.w[,Secondc2pWindowNum.w[,Thirdc2pWindowNum.w]] ; ↩
                     Current to use

                MUsedc2pWindow
                 ; Returns currently used c2pWindow

                MAddrc2pWindow
                [(c2pWindowNumer.w)] ; Returns address of c2pWindow structure

                MCopyc2pWindow
                MSourcec2pWindow.w,Destc2pWindow.w ; Copy definition-data only

shapes


                MReserveShapes
                [(]NumberOfShapes.w[,ShapeBankToUse.w][)] ; Reserve structure- ↩
                    memory for Shapes

                MAutoCookie
                On/Off ; Autocreation of ByteForByte cookies

                MAutoUseShapes
                True/False ; Automatically 'use' new shapes. <>0=True

                MAutoShapeWrap
                On/Off ; Auto X&Y Handle-wrapping for Shapes

                MAutoShapeClip
                Status.b ; Auto-clip new Shapes. On/Off

                MShape
                [(]ShapeNumber.w,Width.w,Height.w[)] ; Allocmem for shape data

                MCludgeShape
                ShapeNumber.w,Width.w,Height.w,Memory.l ; Cludge shape from  ↩
                    existing mem
```

```
MCludgeShapeStruct
[()SourceShape.w,DestShape.w[)] ; Copy definition-data only

MShapeWindow
[()SourceShape.w,DestShape.w,X.w,Y.w,Width.w,Height.w[)] ; Cludge ↩
    Shape within a Shape

MBitmapShape
[()SourceBitmap.w,DestShape.w[)] ; Copy definition-data only

MCludgeCookie
ShapeNumber.w,Memory.l ; Cludge shape's cookie from existing mem

MFreeShape
[FirstShape.w[,LastShape.w]] ; Free a Shape, range of shapes, or  ↩
   all shapes

MShapeWidth
[(ShapeNumber.w)] ; Returns width of Shape

MShapeHeight
[(ShapeNumber.w)] ; Returns height of Shape

MAddrShape
[(ShapeNumber.w)] ; Returns address of Shape structure

MHandle
ShapeNumber.w,XOffset.w,YOffset.w ; Set handle of Shape

MUseShape
MainShapeNum.w[,SecondShapeNum.w[,ThirdShapeNum.w]] ; Current  ↩
   Shape(s) to use

MUsedShape
 ; Returns currently used Shape

MMakeCookie
[FirstShape.w[,LastShape.w]] ; Make a cookie for a shape, range  ↩
   of shapes, or all shapes

MFreeCookie
[FirstShape.w[,LastShape.w]] ; Free a Shape's cookie, a range of  ↩
   Shape's cookies, or all cookies

MShapeWrap
ShapeNumber.w,On/Off ; De/Activate X&Y Handle-Wrap for Shape

MAutoCookieXFlip
On/Off ; Auto X-Flip for Shape's cookie

MAutoCookieYFlip
On/Off ; Auto Y-Flip for Shape's cookie

MAutoCookieFlip
On/Off ; Auto X&Y Cookie-Flip for Shapes
```

```
MShapeXFlip
ShapeNumber.w ; Horizontally flip a Shape (see MAutoCookieFlip)

MShapeYFlip
ShapeNumber.w ; Vertically flip a Shape (see MAutoCookieFlip)

MCookieXFlip
ShapeNumber.w ; Horizontally flip a Shape's cookie

MCookieYFlip
ShapeNumber.w ; Vertically flip a Shape's cookie

MShapeClip
ShapeNumber.w[,X.w,Y.w,Width.w,Height.w][,Active] ; Active=On/Off ↩
    . Define Shape's clip window

MGetaShape
ShapeNumber.w,X.w,Y.w,Width.w,Height.w[,Block?,StencilIsCookie?]  ↩
    ; Grab shape from bitmap

MUseShapeBank
BankNumber.w ; Current shape bank, 0..31

MShapePtr
[Xpos.w,Ypos.w][,ShapeNumber.w] ; Return data address calculated  ↩
    using shape [and coords]

MCookiePtr
[Xpos.w,Ypos.w][,ShapeNumber.w] ; Return address calculated using ↩
     cookie [and coords]
```

bitmaps

```
MReserveBitmaps
[()NumberOfBitmaps.w[)] ; Reserve structure-memory for Bitmaps

MAutoStencil
On/Off ; Autocreation of ByteForByte stencils

MAutoUseBitmaps
True/False ; Automatically 'use' new bitmaps. <>0=True

MAutoBitmapWrap
On/Off ; Auto X&Y Handle-Wrapingp for Bitmaps

MAutoBitmapClip
Status.b ; Auto-clip new Bitmaps. On/Off

MBitmap
[()BitmapNumber.w,Width.w,Height.w[)] ; Allocmem for bitmap data

MCludgeBitmap
BitmapNumber.w,Width.w,Height.w,Memory.l ; Cludge bitmap from  ↩
    existing mem

MCludgeBitmapStruct
```

```
[()SourceBitmap.w,DestBitmap.w[)] ; Copy definition-data only

MBitmapWindow
[()SourceBitmap.w,DestBitmap.w,X.w,Y.w,Width.w,Height.w[)] ;  ↩
   Cludge Bitmap within a Bitmap

MShapesBitmap
[()SourceShape.w,DestBitmap.w[)] ; Copy definition-data only

MCludgeStencil
BitmapNumber.w,Memory.l ; Cludge bitmap's stencil from existing  ↩
   mem

MFreeBitmap
[FirstBitmap.w[,LastBitmap.w]] ; Free a Bitmap, range of bitmaps, ↩
    or all bitmaps

MBitmapWidth
[(BitmapNumber.w)] ; Returns width of Bitmap

MBitmapHeight
[(BitmapNumber.w)] ; Returns height of Bitmap

MAddrBitmap
[(BitmapNumber.w)] ; Returns address of Bitmap structure

MBitmapOrigin
BitmapNumber.w,XOffset.w,YOffset.w ; Set origin of Bitmap

MUseBitmap
MainBitmapNum.w[,SecondBitmapNum.w[,ThirdBitmapNum.w]] ; Current  ↩
   Bitmap to use

MUsedBitmap
; Returns currently used Bitmap

MMakeStencil
[FirstBitmap.w[,LastBitmap.w]] ; Make a stencil for a bitmap,  ↩
   range of bitmaps, or all bitmaps

MFreeStencil
[FirstBitmap.w[,LastBitmap.w]] ; Free's a Bitmap's stencil, a  ↩
   range of stencils or all stencils

MBitmapWrap
BitmapNumber.w,On/Off ; De/Activate X&Y Handle-Wrap for Bitmap

MAutoStencilXFlip
On/Off ; Auto X-Flip for Bitmap's stencil

MAutoStencilYFlip
On/Off ; Auto Y-Flip for Bitmap's stencil

MAutoStencilFlip
On/Off ; Auto X&Y Stencil-Flip for Bitmaps

MBitmapXFlip
```

```
                BitmapNumber.w ; Horizontally flip a Bitmap (see MAutoStencilFlip ↩
                    )

                MBitmapYFlip
                BitmapNumber.w ; Vertically flip a Bitmap (see MAutoStencilFlip)

                MStencilXFlip
                BitmapNumber.w ; Horizontally flip a Bitmap's stencil

                MStencilYFlip
                BitmapNumber.w ; Vertically flip a Bitmap's stencil

                MBitmapClip
                BitmapNumber.w[,X.w,Y.w,Width.w,Height.w][,Active] ; Active=On/ ↩
                    Off. Define Bitmap's clip window

                MGetaBitmap
                BitmapNumber.w,X.w,Y.w,Width.w,Height.w[,Block?,CookieIsStencil?] ↩
                     ; Grab bitmap from shape

                MBitmapPtr
                [Xpos.w,Ypos.w][,BitmapNumber.w] ; Return data address calculated ↩
                     using bitmap [and coords]

                MStencilPtr
                [Xpos.w,Ypos.w][,BitmapNumber.w] ; Return address calculated  ↩
                    using stencil [and coords]

scrolls


                MScroll
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
                    CustomOffsets.l]] ; Copy graphic

                MScrollShape
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
                    CustomOffsets.l]] ; Copy graphic

                MScrollStencil
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
                    CustomOffsets.l]] ; Copy sten to sten

                MScrollCookie
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
                    CustomOffsets.l]] ;Copy cook to cook

                MScrollBitmapToShape
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
                    CustomOffsets.l]];bitmap 2 shape

                MScrollShapeToBitmap
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
                    CustomOffsets.l]] ;shape 2 bitmap

                MScrollStencilToCookie
```

```
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
    CustomOffsets.l]]; sten2cookie

MScrollCookieToStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]] ; cookie2sten

MMaskScroll
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Copy  ↩
    bitmap graphic with stencil-cut

MMaskScrollShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy  ↩
    shape graphic with cookie-cut

MMaskScrollStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w];Copy  ↩
    stencil2stencil & stencil-cut

MMaskScrollCookie
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy  ↩
    cookie to cookie & cookie-cut

MMaskScrollBitmapToShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Copy  ↩
    bitmap to shape & cut

MMaskScrollShapeToBitmap
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy  ↩
    shape to bitmap & cut

MMaskScrollStencilToCookie
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;Copy  ↩
    stencil2cookie & cut

MMaskScrollCookieToStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy  ↩
    cookie2stencil & cut

MBlockScroll
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;  ↩
    BlockCopy graphic

MBlockScrollShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    BlockCopy graphic

MBlockScrollStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;  ↩
    BlockCopy stencil to stencil

MBlockScrollCookie
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    BlockCopy cookie to cookie

MBlockScrollBitmapToShape
```

```
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;  ↩
    BlockCopy bitmap to shape

MBlockScrollShapeToBitmap
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    BlockCopy shape to bitmap

MBlockScrollStencilToCookie
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w]; ↩
    BlockCopy stencil2cookie

MBlockScrollCookieToStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    BlockCopy cookie2stencil

MSScroll
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
    CustomOffsets.l]] ; Copy bm 2 bm and st 2 st

MSScrollShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]] ; Copy sh2sh and ck2ck

MSScrollBitmapToShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
    CustomOffsets.l]];bm2shandst2ck

MSScrollShapeToBitmap
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]]; sh2bmandck2st

MSMaskScroll
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;  ↩
    Stencil-Copy bm 2 bm and st 2 st

MSMaskScrollShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Cookie- ↩
    Copy sh2sh and ck2ck

MSMaskScrollBitmapToShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;Sten- ↩
    Copy bm2sh&st2ck

MSMaskScrollShapeToBitmap
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Cook- ↩
    Copy sh2bm&ck2st

MSBlockScroll
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Block- ↩
    Copy bm 2 bm and st 2 st

MSBlockScrollShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Block- ↩
    Copy sh2sh and ck2ck

MSBlockScrollBitmapToShape
```

```
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w]; ↩
                    BlockCopy bm2sh&st2ck

                MSBlockScrollShapeToBitmap
                X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w];BlockCopy ↩
                     sh2bm&ck2st

                MSScrollCut
                On/Off ; 0=Paste stencil/cookie, <>0=Cut using stencil/cookie

                MMaskScrollMode
                [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
                    MColourMode/MReMapMode/MSimpleReMapMode

                MSMaskScrollMode
                [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
                    MColourMode/MReMapMode/MSimpleReMapMode

drawing


                MDrawingMode
                [()Mode.w()] ; InvMode/MColourMode/MReMapMode/MSimpleReMapMode to ↩
                     use for drawing (MPlot etc)

                MColourMode
                ;Returns value 4 which represents 'colour' mode in the blit modes

                MReMapMode
                ;Returns value 5 which represents 'ReMap' mode in the blit modes  ↩
                    (uses current 2-dimensional table)

                MSimpleReMapMode
                ;Returns value 6 which is 'SimpleReMap' mode in blit modes (uses  ↩
                    current 1-dimensional table)

                MAddMode
                ; Returns value 7 which represents 'add' mode in the blit modes

                MInk
                MainColour.b[,SecondColour.b[,ThirdColour.b]] ; Set what colour  ↩
                    to assume as currently used. 0..255

                MCls
                [Colour] Clear a bitmap to colour 0 or the specified colour

                MClsShape
                [Colour] Clear a shape to colour 0 or the specified colour

                MClsStencil
                [Colour] Clear a stencil to colour 0 or the specified colour

                MClsCookie
                [Colour] Clear a cookie to colour 0 or the specified colour

                MPlot
```

```
Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the bitmap [to the specified colour]

MPlotShape
Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the shape [to the specified colour]

MPlotStencil
Xpos.w,Ypos.w[,Colour] ;Plot a single pixel in the stencil to * represent* the [specified] colour

MPlotCookie
Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the cookie to * represent* the [specified] colour

MPoint
(Xpos.w,Ypos.w[,BitmapToRead.w]) ; Return the colour of a single pixel in a bitmap

MPointShape
(Xpos.w,Ypos.w[,ShapeToRead.w]) ; Return the colour of a single pixel in a shape

MPointStencil
(Xpos.w,Ypos.w[,BitmapToRead.w]) ; Return the status of a single pixel in a stencil. -1=Data, 0=Background

MPointCookie
(Xpos.w,Ypos.w[,ShapeToRead.w]) ; Return the status of a single pixel in a cookie. -1=Data, 0=Background

MBoxF
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a bitmap [to specified colour]

MBoxFShape
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a shape [to specified colour]

MBoxFStencil
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a stencil [to specified colour]

MBoxFCookie
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a cookie [to specified colour]

MBox
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a bitmap [to specified colour]

MBoxShape
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a shape [to specified colour]

MBoxStencil
```

```
Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a  ↩
    stencil [to specified colour]

MBoxCookie
Xpos.w,Ypos.w,Width.w,Height.w[,Colour] Draw an unfilled box in a ↩
     cookie [to specified colour]

MLine
[Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b] ;Draw a line from X1, ↩
    Y1 to X2,Y2 in a Bitmap [in Colour]

MLineShape
[Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b] ;Draw a line from X1, ↩
    Y1 to X2,Y2 in a Shape, [in Colour]

MLineStencil
[Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b];Draw a line from X1,Y1 ↩
     to X2,Y2 in a stencil, [in Col]

MLineCookie
[Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b] ;Draw a line from X1, ↩
    Y1 to X2,Y2 in a cookie, [in Col]
```

tiles

```
MTile16x16
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape to bitmap, ↩
     size must be 16x16, align x/y

MTile32x32
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape to bitmap, ↩
     size must be 32x32, align x/y

MSTile16x16
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape&cook 2  ↩
    bitmap, size 16x16, align x/y

MSTile32x32
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape&cook 2  ↩
    bitmap, size 32x32, align x/y

MSTile16x16Store
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape&cook 2  ↩
    bitmaps, size 16x16, align x/y

MSTile32x32Store
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape&cook 2  ↩
    bitmaps, size 32x32, align x/y

MTile16x16Store
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape to 2  ↩
    bitmaps, size 16x16, align x/y

MTile32x32Store
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape to 2  ↩
    bitmaps, size 32x32, align x/y
```

queues

```
MReserveQueues
[(]NumberOfQueues.w[)] ; Reserve structure-memory for Queues

MAutoUseQueues
True/False ; Automatically 'use' new Queues. <>0=True

MQueue
[(]QueueNumber.w,NumberOfItems.w[)] ; Allocmem for Queue list items

MQDummy
[Queue.w,]Xpos.w,Ypos.w,Width.w,Height.w ; Add an item to a queue without having to do a blit

MUnQueue
QueueNumber.w[,FirstItem.w,LastItem.w][,BitmapNumber.w] ; UnQueue [range of] queued objects [&flush]

MBlockUnQueue
QueueNumber.w[,FirstItem.w,LastItem.w][,BitmapNumber.w];Block- UnQueue [range of] objects[&flush]

MFlushQueue
QueueNumber.w ; Empties the queue to contain no items

MFreeQueue
[FirstQueue.w[,LastQueue.w]] ; Free a Queue, a range of queues, or all queues

MAddrQueue
[(QueueNumber.w)] ; Returns address of Queue structure

MUseQueue
MainQueueNum.w[,SecondQueueNum.w[,ThirdQueueNum.w]] ; Current to use

MUsedQueue
 ; Returns currently used Queue
```

blits

```
MBlitMode
[([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ MColourMode/MReMapMode/MSimpleReMapMode

MBlit
[ShapeNumber.w,]Xpos.w,Ypos,w ; Blit shape to bitmap, any coords

MBlock
[ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit shape to bitmap, align Xpos and width in multiples of 16!
```

```
                MQBlitMode
                [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
                   MColourMode/MReMapMode/MSimpleReMapMode

                MQBlit
                [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos,w ; QBlit shape to bitmap,  ↩
                   any coords

                MQBlock
                [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos.w ; QBlock-blit shape 2  ↩
                   bitmap, align Xpos & width in mult of 16

                MSBlitMode
                [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
                   MColourMode/MReMapMode/MSimpleReMapMode

                MSBlit
                [ShapeNumber.w,]Xpos.w,Ypos,w ; Blit shape to bitmap and cookie  ↩
                   to stencil, any coords

                MSBlock
                [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit shape to bitmap &  ↩
                   cookie 2 stencil, Xpos&Width in 16's

                MSBlitCut
                On/Off ; 0=Paste stencil/cookie, <>0=Cut using stencil/cookie

                MQSBlitMode
                [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
                   MColourMode/MReMapMode/MSimpleReMapMode

                MQSBlit
                [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos,w ; QBlit shape to bitmap  ↩
                   and cookie to stencil, any coords

                MQSBlock
                [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos.w ; QBlock-blit shape 2  ↩
                   bitmap, Xpos&width mult of 16

                MQSBlitCut
                On/Off ; 0=Paste stencil/cookie, <>0=Cut using stencil/cookie.  ↩
                   Adds entry to queue

tables


                MReserveTables
                [()NumberOfTables.w[)] ; Reserve structure-memory for Tables

                MAutoUseTables
                True/False ; Automatically 'use' new Tables. <>0=True

                MTable
                [()TableNumber.w,SizeInBytes.l[)] ; Allocmem for Table list items

                MFreeTable
```

```
                [FirstTable.w[,LastTable.w]] TableNumber.w ; Free a Table, a  ↩
                   range of tables or all tables

                MAddrTable
                [(TableNumber.w)] ; Returns address of Table structure

                MUseTable
                MainTableNum.w[,SecondTableNum.w[,ThirdTableNum.w]] ; Current to  ↩
                   use

                MUsedTable
                ; Returns currently used Table

                MTablePtr
                [TableNum.w] ; Returns pointer to base of the table itself

transfer


                MPlanar16ToBitmap
                BitmapNum.w,PlanarAddr.l[,OpWidth.w,OpHeight.w,PlanarWidth.w, ↩
                   PlanarHeight.w] ; Convert p2c

                MPlanar16ToShape
                ShapeNum.w,PlanarAddr.l[,OpWidth.w,OpHeight.w,PlanarWidth.w, ↩
                   PlanarHeight.w] ; Convert p2c

                MReMap
                [Colour#0.b,Colour#1.b,BitmapNum.w] *or* [RemapTable.l[,BitmapNum ↩
                   .w]] ; Remap #0 to #1 or with table

                MReMapShape
                [Colour#0.b,Colour#1.b,ShapeNum.w] *or* [RemapTable.l[,ShapeNum.w ↩
                   ]] ; Remap #0 to #1 or with table

                MPictureDissolveIn
                PictureBitmapNum.w,Colour.b ; Do a picture-based colour-number  ↩
                   dissolve-in of a bitmap

                MPictureDissolveOut
                PictureBitmapNum.w,Colour.b,WipeToColour.b ;Do picture-based  ↩
                   colour dissolve-out of bitmap

                MZoom
                SrcX.q,SrcY.q,XAdd.q,YAdd.q,DestX.w,DestY.w,OpWidth.w,OpHeight.w, ↩
                   DeRes?.w[,SrcBmap.w[,CustomOffsets.l]]

                MZoomShape
                SrcX.q,SrcY.q,XAdd.q,YAdd.q,DestX.w,DestY.w,OpWidth.w,OpHeight.w, ↩
                   DeRes?.w[,SrcShap.w[,CustOffs.l]]

                MZoomBitmapToShape
                SrcX.q,SrcY.q,XAdd.q,YAdd.q,DstX.w,DstY.w,OpWid.w,OpHeight.w, ↩
                   DeRes?.w[,SrcBmap.w[,CustOffs.l]]

                MZoomShapeToBitmap
```

```
                SrcX.q,SrcY.q,XAdd.q,YAdd.q,DstX.w,DstY.w,OpWidth.w,OpHeight.w, ↩
                   DeRes?.w[,SrcShap.w[,CustOffs.l]]

particles


                MParticleMode
                [([]Mode.w[])] ; MColourMode, MSimpleReMapMode or MReMapMode - to ↩
                     use in particle plot/draw

                MParticleFormat
                [(]Format.b[)] ; Set particle lists/operation format. 0 = X.w,Y.w ↩
                   , <0 = X.q,Y.q, >0 = Ptr.l

                MPlotParticles
                CoordinateList.l,NumPoints.l[,Colour.b] ; Plot lots of points  ↩
                   from a table of positions

                MGrabParticles
                CoordinateList.l,NumPoints.l,Buffer.l ; Grab lots of points from  ↩
                   a table into buffer mem

                MDrawParticles
                CoordinateList.l,NumPoints.l,Buffer.l ; Draw lots of previously  ↩
                   grabbed points using a table

                MGrabParticlesAndPlot
                CoordinateList.l,NumPoints.l,Buffer.l[,Colour.b]; Grabs points to ↩
                     buffer & plots table

                MAddToParticles
                CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add [two sets of]  ↩
                   increments to particle list

                MWrapParticles
                CoordinateList.l,NumPoints.l ; Bring particles in from opposite  ↩
                   edge to which they left

                MReboundParticles
                CoordinateList.l,NumPoints.l,DirectionList.l,DetectSize.w ;  ↩
                   Bounce off edges (NOT Ptr.l!!!)

                MAddXYToParticles
                CoordinateList.l,NumPoints.l,XToAdd.w,YToAdd.w ; Add constants to ↩
                     all particles

                MAddXYToParticlesA
                CoordinateList.l,NumPoints.l,ValueToAdd.l ; Add constant to all  ↩
                   particle pointers

                MAddXYToParticlesQ
                CoordinateList.l,NumPoints.l,XToAdd.q,YToAdd.q ; Add constants to ↩
                     all particles

                MWrapXParticles
                CoordinateList.l,NumPoints.l ; Bring particles in from left/right ↩
                     edges (Not Ptr.l)
```

```
                MWrapYParticles
                CoordinateList.l,NumPoints.l ; Bring particles in from top/bottom ↩
                    edges

                MAddToXParticles
                CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add X components  ↩
                   of [two sets of] increments

                MAddToYParticles
                CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add Y components  ↩
                   of [two sets of] increments

others


                MGenericPtr
                Xpos.w,Ypos.w,BaseAddress.l,RowWidth.w ; Calculate and return  ↩
                   address based on inputs

                MMildredBase
                ; Returns long address of the base of Mildred's internal data  ↩
                   area
```

1.2 Mildred information

Mildred is a chunkygraphics library for Blitz Basic 2, providing extensive chunky-format graphics capabilities for your Blitz software. The library, its documentation and accompanying programs are Copyright (c) 1998-1999 Paul West.

Copyright has been reclaimed from Pagan Games and is now owned fully by the author, Paul Jonathan West.

Please refer to the archive containing the old on-line html documentation in order to obtain an in-depth manual regarding specifically the chunky-to-planar related commands from the library.

Primarily, have runtime error-checking ON while figuring out how things work as there is extensive checking performed throughout the system and the messages will help you to avoid crashes and to supply the correct parameters. Once the program is running well and won't fall over, turn runtime error-checking off. Your final executable should (highly recommended) be compiled without runtime errorchecking, as the runtime errorchecking routines do gobble up a fair amount of time, especially if you are calling a lot of small routines like plotting lots of pixels.

In Mildred, shapes and bitmaps are internally identical, which also means that bitmaps have stencils in the same way that shapes have cookies. Also note that ALL Mildred commands begin with the identifying letter 'M'. There are 32 shape banks. 'Current' shapes refer to the currently used bank, and operations accross banks are not possible.

As of v1.23, space is reserved for default amounts of objects so that for most smaller projects you do not need to use the MReserveXXX instructions. By default,

there are 20 c2pWindows, 20 chunky Bitmaps, 100 chunky Shapes and 20 Queues. If you want to use more than this you should reserve some specifically. The objects that are reserved by default consume less than 10k of fastram each time your program is run.

In v1.27 the ordering of lookup tables for the MReMapUsingShape and MReMapShapeUsingShape commands has been reversed. This is to provide consistency with the new MReMap blit mode, which makes it possible to reduce the size of tables when a shape contains a lower number of colours (starting from 0). Also in v1.27 a default amount of 20 chunky Table objects will be allocated at runtime.

As of v1.36 the lib will automatically check for what cpu is available from ExecBase\AttnFlags and will make automatic calls to MCPU, Mc2pCPUmode and M040c2pUsage. MCPU will be set to whatever cpu is detected, and this will serve as the default unless you specifically use MCPU in your program. Mc2pCPUmode will be set to a default that best suits the detected cpu. If an 040 or higher is not present, M040c2pUsage will be switched off. M040c2pUsage overrides Mc2pCPUmode and it used to be possible to assume that the 040Usage was always on. This assumption is no longer valid but hopefully if you use MProcessor or Processor to return a value to pass as the cpu to use, it will be in accordance with the value that was internally detected so if an 040 is available it shouldn’t have switched of the M040c2pUsage. The end result should hopefully be the same for the programmer unless you attempt to ask for a specific CPU that may not equate to the cpu that is actually available, in which case care must be taken to switch M040c2pUsage on if necessary.

Also as of v1.36 the MProcessor function has been included to remove reliable on the old blitz ‘Processor’ instruction and it is recommended that this be used. Values returned range from 0 to 6 rather than 0..4, although the lib will mostly assume that anything higher than 4 means to use the 040+-only routines.

As of v1.38 some previously created tokens were removed and merged in with others to provide a simpler interface and reduce the number of tokens in the library. Mainly the only tokens affected are the particle-animation ones, but also MProcessor. If you had used these tokens prior to version 1.38, it is recommended that you save your program as ascii text, THEN install the new version of Mildred, and then reload your ascii-format program into blitz so that it tokenises properly. If you attempt to save off the ascii text after installing the library the tokens would be corrupt when you tried to load it back in. If unsure, keep a temporary backup of your old Mildred.obj library file. Note that some tokens have been removed, such as MAddXToParticles[Q] and MAddYToParticles[Q], with their functionality being implemented into MAddXYToParticles[Q] as optimised routines to use when XToAdd or YToAdd are 0. Also the MAdd2To.. family of commands have been merged into the originally single MAddTo.. family which now allows you to specify a second increment list as an optional parameter.

In v1.39 library base passing was implemented so that other blitz libraries can obtain the base of Mildred’s internal data area as part of the !libs macro in their library definition. In the event that this doesn’t work, a function called MMildredBase has been added to return the same address of the base of the internal data area, which you can feed to your library with a token designed for that purpose.

As of v1.42 it has become more efficient to use shapes or define scroll widths that are multiples of 16, as specially optimised routines can then be used. This

does not mean that the positioning of blits or scrolls is affected in the way that a block[scroll] would be, or that you have to restrain your widths to multiples of 16 compulsorarily. What it does mean is that if you can live with widths that are a multiple of 16 the blits will be a little faster. Also the blockscrolls will be slightly faster if the widths are multiple of 64.

IMPORTANT: In v1.43 a number of important changes have taken place to the token structure, the number of tokens, their syntax and functionality. Therefore if you install this version or later you will need first to save (all?) of your Mildred programs as ascii text. Then you can install v1.43, and when you reload your ascii programs they should tokenise properly. Be careful about doing this because if you cause your program to be tokenised wrongly the errors will be widespread. A number of things have changed in order to tidy up the library and prepare for the future. A similar but smaller-scale alteration was made in v1.38. Once you've saved your programs as ascii, installed the new library version, and loaded back in your ascii texts, there may be a few small manual changes you will need to make!

Importantly, almost all of the particle tokens have collaged into less tokens. The word, quick and address variations have been put into single tokens, and the selection of the particle format (X.w,Y.w, or X.q,Y.q or Ptr.l) is now made by a new token called MParticleFormat. So you will need to add MParticleFormat somewhere in your program or your particle stuff will likely not work properly (ie use the wrong format). The only particle tokens that aren't changed are MAddXYToParticles[A/Q] because they have to have different variable types as input. If you had previously used 'Q' or 'A' particle commands in your program they will likely no longer tokenise. You need to go through your program and remove the Q or A from the end of the token names!

In 1.43 a few tokens have been scrapped altogether. MReMapUsingShape has been completely removed. Its functionality can be implemented using a remapped blit or scroll. Also MReMapShapeUsingShape as been scrapped and its functionality can again be replaced with a scroll. Mc2pToggleSingle, Mc2pToggleDouble and Mc2pToggleTriple have been removed. Instead you should use Mc2pToggle, to which you need to specify wether you're using single, double, or triplebuffering. MMidHandle and MBitmapMidOrigin have been removed. Their functionality can be easily replaced with blitz code such as MHandle MShapeWidth/2,MShapeHeight/2. MCopyHandle and MCopyOrigin have also been scrapped. I assume that if you are able to set one handle you can set others also. M040c2pUsage has been scrapped and everything relating to it. Now Mc2pCPUmode is the command to use to select a specific c2p routine and the routines available will not be limited even if an 040 cpu is not available (it will not crash, just be less efficient). MFlushTable has been deleted as it wasn't really doing anything anyway and was leftover from cut-and-paste of most of the queue system when making the table commands.

Now, some important changes were also made to the MUse and MFree type commands, such as MUseBitmap, MUsec2pWindows, etc. The 'plural' versions have been scrapped and their functionality has been placed in the 'singular' versions. So you would now use MUseBitmap in order to specify one, two or three currently-used bitmaps. You can also still specify a 'range' of objects. So if you had used multiple-operation commands such as MFreeShapes or MUseBitmaps in your programs you will find that they need to be changed to the singular version by removing the 's' from the end of the token names, if they tokenise to the old names at all. If you want to free all objects of a particular type you would now just use MFreec2pWindow for example, maybe MFreeCookie, without any parameters. Finally, the same sort of modification has been made to MMakeCookies and MMakeStencils. Their functionality has been moved to the singular versions MMakeCookie and

MMakeStencil. It is again still possible to make a range of or even all cookies/stencils using two or no parameters. If you've used the plural versions of these tokens in your programs you will need to change them.

BEWARE that tokens which have been removed from the library will, when you load in your saved ascii sourcecode probably tokenise to something else entirely, or due to the number of overall tokens having been decreased by about 40 or so you may get some ?????'s in your code. I suggest you cross-refernce your ascii sourcecode to find out what the tokens were and take the appropriate action to replace them with legal equivelents.

I am sorry that these changes to the library will cause this extra manual work for you but I've done pretty much everything I wanted to do all at once to save an ongoing hassle, and now that there is quite a lot of room for new tokens it will be possible to provide extra and better enhanced features in future updates.

In v1.43 it seemed that the 128th token, which was MPictureDissolveIn, was not working. It kept reporting syntax error even with correct syntax, and also tended to remove any parameters typed in. To rectify this in v1.44 I renamed the 128th token to MNothing, which does nothing, and have moved MPictureDissolveIn to the end of the library. If you've used MPictureDissolveIn in your programs you will probably need to change it from being tokenised as MNothing.

In v1.45 MUnQueueRange has been removed and its functionality put into MUnQueue as extra syntax. The rule that unqueueing a range of items does not flush the queue still applies. MBlockUnQueue has been added in place of MUnQueueRange, to unqueue block areas, but it doesn't check to see that queue entries are valid so that is up to you. Widths should be multiple of 16 and unless MCPU is set to 3 or lower, the X coordinate should be a multilpe of 16 also. That it is in place of MUnQueueRange means that any MUnQueueRange commands in your program would now tokenise as MBlockUnQueue, so may need changing to MUnQueue. Also MCludgeShape, MCludgeBitmap, MCludgeCookie and MCludgeStencil have been 'fixed' to take the base address you pass to it as absolute, rather than requiring you to do a '-16' in your code. You should still ensure that the memory is aligned to a multiple of 16.

Importantly in v1.45 all c2p interlacing code has been axed. This includes 5 tokens, namely Mc2pRowLacing, Mc2pColumnLacing, Mc2pRowToggle, Mc2pColumnToggle and Mc2pToggle. This does not affect tokenisation but if you used these commands before you will just have to remove them. I think the interlacing was useless and had no real application. Taking this out slightly speeds up the c2p operation. To facilitate this change, some of the internal data area has been altered and c2pWindow structures have been reduced to 8 bytes instead of 16, with support for the new non-modulo c2p routines which are selected automatically if your c2pWindow has no linemodulos for slightly improved performance.

In v1.48 I have added a DeRes?.w parameter as the last compulsory parameterof MZoom, MZoomShape, MZoomBitmapToShape and MZoomShapeToBitmap. This means that if you had used the optional BitmapNum.w or CustomOffsets.l parameters you will need to insert a parameter prior to these. Normal zooms should have a DeRes?.w parameter of False or 0, or to perform a De-Resolution zoom you should make it True or <>0. Note also there are highly optimised routines for when widths are multilpes of 4, 16, when the adders are integers without decimal parts, and when the xadder is 1.0 (for a y-zoom).

In v1.50, full copyright and ownership of Mildred was made property of Pagan Games. This is no longer valid. Mildred is Copyright (c) 1998-1999 Paul West and

I own full rights.

1.3 Library History

This is an account of the library history since its release.

```
v1.1  - First public release
v1.11 - Fixed bug in MUseShapeBank and altered ShapesTotal size to word as it was
   incorrectly a longword
      - Fixed bug in the errorchecking of Mc2pCPUmode that was checking d3 instead
        of d0
v1.12 - Fixed bug in MUnQueue that would only do two lines of code if wrapping was
    active, but should have been done always
v1.13 - MBoxF, MBoxFShape, MBoxFStencil and MBoxFCookie added
      - MBox, MBoxShape, MBoxStencil and MBoxCookie added
v1.14 - MPlanar16ToBitmap, MPlanar16ToShape added
v1.15 - Planar-to-chunky converter optimised further using addx and reverse
   bitplane order, twice as fast as roxr.b #n,dn
      - MGenericPtr added
      - A shape's handle is unconditionally added (actually subtracted) to Xpos,
        Ypos in shape-to-bitmap type blits (MBlit etc)
v1.16 - Fixed small bug in !PerformPoint macro, d6.l should have been d6.w.
      - Fixed small but ineffective bug in MPlotCookie, d6.l should have been d6.w
v1.17 - Added MCludgeCookie and MCludgeStencil, also needed to add two macros
      - Fixed bug in macro used by MCludgeCookie and MCludgeStencil, as it was not
        setting 'SHere' to 0 to indicate cludge.
v1.18 - Added MUnQueueRange for unqueuing a range of items and without flushing
   the queue
v1.19 - Fixed bug in data for shape banks, was using structures of 8 bytes but
   only were 6 bytes in mem
      - Fixed bug in MReserveShapes, was shifting bank number 8 places instead of
        3
v1.2  - Added MReMap and MReMapShape
v1.21 - Fixed bug in macro DeallocStencil, was killing the whole object
v1.22 - Commented-out line in MShapeClip and MBitmapClip to make X leftedge
   unaligned (width is still multiple of 4)
      - Commented-out line in Macro CludgeResourceWindow to make X leftedge
        unaligned (width of window is still multiple of 4)
v1.23 - Added mode to MBlit so that if a cookie is not present it will just blit
   the graphic in 'replace'-mode (unmasked)
      - Minor pipeline improvement in macros !PerformPlot and !PerformPoint
      - Added code to the init routine to reserve default amounts of all objects
        at runtime (doesn't need much mem)
      - Added 'BankToUse' parameter to MReserveShapes so that you don't have to do
        a seperate MUseShapeBank
      - Changed BoxF and Box routines to use X2,Y2 instead of Width,Height and had
        to add macro CCheckWindowFits4
v1.24 - Made the colour parameter in MPlot,MPlotShape,MPlotStencil and MPlotCookie
    optional, assuming 0 if not specified
      - *Partially* Added MLine, MLineShape, MLineStencil and MLineCookie
      - Modified shapebank-related routines to provide 32 shape banks numbered
        0..31, instead of 0..9.
      - Fixed bugs in macros ShuffleRegs1, ShuffleRegs2 and ShuffleRegs3
v1.25 - Added MReMapUsingShape and MReMapShapeUsingShape
v1.26 - Optimised routine PerformReMapUsing for slight speed gain
```

```
      - Optimised routine PerformReMap for 25% speed gain in table mode
      - Fixed bug in macro CCheckWindowFits4, which affected runtime errorchecking ↩
        of MBox/MBoxF and related routines
v1.27 - Added MInk for setting a currently-used pen colour. Defaults to 1 which is ↩
    a bit more logical than 0
      - Added MColourMode function to accompany CookieMode/SolidMode etc, but for  ↩
        'colour' drawing mode in blits
      - Added 'colour' mode to MBlit, MQBlit, MSBlit, MQSBlit and MMaskScroll  ↩
        routines
      - Removed redundant instruction in routine PerformBlit3's loop, for cookie- ↩
        mode stencil-blits (slight speedup)
      - Adjusted graphics routines to use the ink colour if assuming which colour  ↩
        to use, rather than 0
      - Further optimised routine PerformReMapUsing for slight speedup (about 1-2 ↩
        fps)
      - Changed order of tables being used for MReMap[Shape]UsingShape, for  ↩
        consistency with 'MReMapMode' blit mode
      - Added support for new Table objects
      - Added MReserveTables, MFreeTables, MFreeTable, MAddrTable, MTable
      - Added MFlushTable, MAutoUseTables, MUseTables, MUseTable, MUsedTable
      - Added MTablePtr, MReMapMode
      - Added new blit mode 'ReMap' to the MBlit, MQBlit, MSBlit, MQSBlit and  ↩
        MMaskScroll routines for table-based remapping
v1.28 - Optimised MPictureDissolveIn for speed gain (a good few fps)
      - Added MSimpleReMapMode
      - Added new blit mode 'SimpleReMap' to the MBlit, MQBlit, MSBlit, MQSBlit  ↩
        and MMaskScroll routines for 1-dim remapping
      - Fixed bug in definitions, MPointStencil and MPointCookie were defined as  ↩
        statements, but should have been functions
      - Changed the !PerformPoint macro to initialise d0 before grabbing the byte, ↩
        in case it causes corrupt return value
v1.29 - Optimised non-cut routine used by MSMaskScrolls (PerformGenericBlit6[b])
      - Slight optimisation to non-cut plain copy routine used by M[Q]SBlits ( ↩
        PerformBlit2)
      - Finished MLine, MLineShape, MLineStencil and MLineCookie
v1.30 - Redirected routine PerformGenericBlit3[b] to use PerformBlit1[b], to save  ↩
   code redundancy, and made gen3b into 1b
      - Added MSMaskScrollMode to support blit modes for MSMaskScrolls (previously ↩
        only MMaskScrolls)
      - Redirected routine PerformGenericBlit6[b] to use PerformBlit2[b], to  ↩
        prepare for shared sblit blit-mode code
      - Redirected routine PerformGenericBlit9[b] to use PerformBlit3[b], to  ↩
        prepare for shared sblit cut blit-mode code
      - Completed support for MSMaskScrolls in 'copy' mode with blit modes, by  ↩
        adding PerformBlit2b (2 backwards)
      - Completed support for MSMaskScrolls in 'cut' mode with blit modes, by  ↩
        adding PerformBlit3b (3 backwards)
      - Modified runtime errorchecking routines for MSMaskScrolls to check that  ↩
        tables are available in M[Simple]ReMapMode
v1.31 - Modified MScroll routines to support any width (non multiple, as low as 1) ↩
   .
      - Modified MSScroll routines to support any width (non multiple, as low as  ↩
        1) in both 'paste' and 'cut' modes.
v1.32 - Fixed bugs in routine PerformBlit2[b] for non-cut output to stencil. Some  ↩
   OR's should have been AND's, and vice versa.
v1.33 - Fixed bug in routine PerformLine, sometimes d5 was plotted rather than d6
```

```
v1.34 - Added MPlotParticles for plotting list of pixels to a colour. List items ↩
   are X.w,Y.w
      - Added MGrabParticles for grabbing list of pixels to a buffer. List items ↩
        are X.w,Y.w. Buffer is Pixel.b's
      - Added MDrawParticles for drawing grabbed list of pixels from a buffer. ↩
        List items are X.w,Y.w. Buffer is Pixel.b's
      - Added MGrabParticlesAndPlot for grabbing and plotting pixels to a colour. ↩
        List items are X.w,Y.w. Buffer is Pixel.b's
      - Fixed bug in initialisation, auto-clip for bitmaps and shapes shouldn't ↩
        have been automatically On!
      - Added MPlotParticlesA, MGrabParticlesA, MDrawParticlesA, ↩
        MGrabParticlesAndPlotA, for actual-address list items
      - Added MPlotParticlesQ, MGrabParticlesQ, MDrawParticlesQ, ↩
        MGrabParticlesAndPlotQ, for X.q,Y.q items [*16*.16][*16*.16]
      - Added MAddToParticles, MAddToParticlesA, MAddToParticlesQ, for adding ↩
        values to particle list items
      - Added MAdd2ToParticles, MAdd2ToParticlesA, MAdd2ToParticlesQ, for more ↩
        efficient multiple adds to list items
v1.35 - Added MWrapParticles, MWrapParticlesA, MWrapParticlesQ, to wrap coords ↩
   around edges of bitmap/clip (within reason)
      - Fixed bugs in clip routine of MAddToParticlesQ and MAdd2ToParticlesQ, ↩
        offsets and adders and adding were wrong
      - Fixed bugs in clip routine of MWrapParticles and MWrapParticlesQ, 2 ↩
        conditional branches to loop missing
v1.36 - Added MReboundParticles and MReboundParticlesQ, for bouncing particless ↩
   off the edges. No 'A' version, not possible
      - Fixed bugs in MLine, MLineShape, MLineStencil, MLineCookie, short version ↩
        used wrong colour
      - Added MProcessor function, to replace blitz's 'Processor' instruction and ↩
        support 060
      - Modified various cpu-related routines (c2p and 040 choices) to support ↩
        possible 060 cpu number
      - Modified init routine to check for cpu availability and set MCPU, ↩
        Mc2pCPUmode and M040c2pUsage to appropriate defaults
v1.37 - Added MAddXYToParticles and MAddXYToParticlesQ for adding X and Y ↩
   constants to X and Y components in a particle list
      - Added MAddXYToParticlesA to add constant value to list of Ptr.l particles
      - Added MAddXToParticles, MAddYToParticles, MAddXToParticlesQ and ↩
        MAddYToParticlesQ for further adding to particle lists
v1.38 - Removed unnecessary code from MPlotParticlesA, MGrabParticlesA, ↩
   MDrawParticlesA and MGrabParticlesAndPlotA
      - Added MParticleMode to choose MColourMode, MReMapMode or MSimpleReMapMode ↩
        for particle plot/draw
      - Added MSimpleReMapMode and MReMapMode support to MPlotParticles, ↩
        MPlotParticlesA and MPlotParticlesQ (clipping also!)
      - Added MSimpleReMapMode and MReMapMode support to MDrawParticles, ↩
        MDrawParticlesA and MDrawParticlesQ (clipping also!)
      - Added MSimpleReMapMode and MReMapMode support to MGrabParticlesAndPlot[A/Q ↩
        ] for remap plot and normal grab (and clip!)
      - Merged MAdd2ToParticles[A/Q] into extension of MAddToParticles[A/Q] to ↩
        make friendlier interface & cut down on tokens
      - Token order has been compromised due to removal of MAdd2ToParticles, ↩
        MAdd2ToParticlesA and MAdd2ToParticlesQ !!!
      - Fixed errornumber bugs in errorchecking routines of MShapePtr and ↩
        MCookiePtr. Was Error28, should have been Error27
      - Added support to MBitmapPtr, MStencilPtr, MShapePtr and MCookiePtr to ↩
        assume currently used objects if no params
```

```
      - Merged MAddXToParticles[Q] into special-case routine of MAddXYToParticles[ ↩
        Q] (called if Y is 0)
      - Merged MAddYToParticles[Q] into special-case routine of MAddXYToParticles[ ↩
        Q] (called if X is 0)
      - Token order has been compromised due to removal of MAddXToParticles[Q] and ↩
         MAddYToParticles[Q] !!!
v1.39 - Optimised routine PerformLine for speedup and less instructions
      - Expanded MInk to allow specification of second and third inks to use
      - Slightly rearranged internal data area and ensured alignment without Even4 ↩
        's, ready for extensions to access
      - Added MMildredBase to return the base address of Mildred's internal data  ↩
        area
      - Added lib support for returning base of Mildred's internal data area to  ↩
        other libraries, at the end of Initialise
v1.40 - Added CRsrc_TotWidth to resource structure, for shapes and bitmaps, and  ↩
   added support code in necessary routines
      - Modified many routines to read CRsrc_TotWidth rather than move.w mem->reg, ↩
         add.w mem->reg. Slight general speedups
      - Rearranged part of Mc2pWindow to check and set auto-use at the start as it ↩
         was possibly trashed by the CacheClearU_()
v1.41 - Made MParticleMode, MBlitMode, MQBlitMode, MSBlitMode, MQSBlitMode,  ↩
   MMaskScrollMode and MSMaskScrollMode into commands
      - Slightly modified MUnQueue, negligible speedup, but better pipelining
      - Added MDrawingMode to change mode used for drawing operations such as  ↩
        MPlot, MLine, MBoxF etc
      - Added MDrawingMode support to MCls family for InvMode, MColourMode,  ↩
        MReMapMode and MSimpleReMapMode (no stencil remap!)
      - Added MDrawingMode support to MPlot family for InvMode, MColourMode,  ↩
        MReMapMode and MSimpleReMapMode (no stencil remap)
      - Added MDrawingMode support to MBoxF family for InvMode, MColourMode,  ↩
        MReMapMode and MSimpleReMapMode (no stencil remap)
      - Added MDrawingMode support to MBox family for InvMode, MColourMode,  ↩
        MReMapMode and MSimpleReMapMode (no stencil remap!)
      - Added MDrawingMode support to MLine family for InvMode, MColourMode,  ↩
        MReMapMode and MSimpleReMapMode (no stencil remap)
v1.42 - Moved _DrawingModeType in data area and made it into a word, as it was  ↩
   mistakenly defined as a byte so was trashing
      - Fixed bugs in MSTile32x32 and MSTile32x32Store, wrapping was not  ↩
        implemented for the dest stencil
      - Slightly optimised routine PerformRemap and fixed bug in  ↩
        PerformRemapUsingShape as top half of d4 needed to be cleared
      - Slightly optimised MUnQueue (bitmap form), speedup of about 0.1fps!
      - Fixed major bug in routine PerformBlit1, ReMap mode routine was doing  ↩
        stencil cut but should have been a straight blit
      - Added optimisation as routines PerformBlit1[b]_16, to do groups of 16  ↩
        pixels instead of 4 if the width is a multiple
      - Made M[c2pWindow][Shape][Bitmap]Width and M[c2pWindow][Shape][Bitmap] ↩
        Height work without specified object number
      - Made MAddr[c2pWindow][Shape][Bitmap][Queue][Table] work without specified  ↩
        object number
      - Optimised macro RemakeStencil, used for generating a stencil or cookie  ↩
        from a bitmap or shape
      - Added optimisation as routines PerformBlit2[b]_16, to do groups of 16  ↩
        pixels instead of 4 if the width is a multiple
      - Fixed major bug in routines PerformBlit2[b][_16] and PerformBlit3[b] in  ↩
        SolidMode write to mask was or instead of and
```

```
      - Recoded all tile blits in 030 mode from movem.l's to move.l's and done ↩
        modulos different. 2-3fps faster, on 040.
      - Added optimisation as routines PerformBlit3[b]_16, to do groups of 16 ↩
        pixels instead of 4 if the width is a multiple
      - Optimised GenericBlit (scroll) routines, optimised for when width is ↩
        multiple of 16 or multilpes of 64 for blockscroll
      - Fixed bug in all tokens that add items to a queue. Upper word of d7 was ↩
        corrupt due to unusual longword usage
v1.43 - Added MParticleFormat to select mode for particle tokens, between 0=word, ↩
   <0=quick and >0=actual memory addresses
      - Merged Word/Quick/Address versions of particle commands into single ↩
        commands, using MParticleFormat to choose
      - Removed MReMap[Shape]UsingShape commands as they have been superceded by ↩
        MDrawingMode functionality
      - Removed Mc2pToggleSingle, Mc2pToggleDouble and Mc2pToggleTriple, as you ↩
        might as well just use Mc2pToggle.
      - Removed the MInitShape token and just kept with MShape for making new ↩
        shapes
      - Renamed MUse[c2pWindows][Shapes][Bitmaps][Queues][Tables] to the singular ↩
        versions and removed the plurals
      - Removed MMidHandle, MMidOrigin, MCopyHandle and MCopyOrigin completely
      - Removed the unused MFlushTable completely
      - Removed M040c2pUsage and any other tokens associated with its status.
      - Merged MMakeCookies into MMakeCookie and removed MMakeCookies token
      - Merged MMakeStencils into MMakeStencil and removed MMakeStencils token
      - Merged MFree[c2pWindows][Shapes][Bitmaps][Cookies][Stencils][Queues][ ↩
        Tables] to the singular and removed the plurals
      - Tokenisation has been severly compromised due to changes, deletions and ↩
        additions.!!
v1.44 - Moved MPictureDissolveIn token to the end of the library and renamed the ↩
   old one to MNothing as the 128th token fails
v1.45 - Killed Mc2pRowLacing, Mc2pColumnLacing, Mc2pRowToggle, Mc2pColumnToggle ↩
   and Mc2pToggle. Tokens 1-5 free for use!
      - Removed all code relating to interlaced c2p conversion, especially from ↩
        Mc2pWindow, Mc2p and data area
      - Changed c2pWindow structure to 8 bytes instead of 16 and allowed ↩
        c2p0_Pixels to be .w or .l depending on modulos>0
      - Modified Mc2pWindow and Mc2p to support non-modulo c2p routine, and ↩
        implemented non-modulo c2p (040 and 030) into Mc2p
      - Added MPictureDissolveOut to wipe image to a colour, bit faster than cross ↩
        -wiping to blank image in MPictureDissolveIn
      - Modified MCludge[Shape][Bitmap][Cookie][Stencil] to take the '-16' away ↩
        from the programmer making base address actual
      - Merged MUnQueueRange into extra syntax option of MUnQueue, still able to ↩
        unqueue range without flush or all with flush
      - Added MBlockUnQueue in place of MUnQueueRange for unqueing items with ↩
        width and X coord multiple of 16 (doesn't check)
      - Modified clearscreen mode of MUnQueue and MBlockUnQueue to use current ink ↩
         instead of just 0's
      - Tried to add stencil-unqueue (cut behind stencil) but failed, not enough ↩
        regs, so removed
      - Slightly optimised clearscreen mode of M[Block]UnQueue to do forwards ↩
        operation and (an)+ (removes need for mulu)
      - Slightly optimised routine PerformCls in all relevant blit modes
v1.46 - Added MWrapXParticles and MWrapYParticles to do wrapping of only sides or ↩
   top/bottom. No X-wrap for Ptr.l, however.
```

```
      - Added MAddToXParticles and MAddToYParticles to add list(s) to particles ↩
        but only one of the two components (Not Ptr.l)
      - Fixed small bug in errorchecking routine of MAddToParticles, check for ↩
        zero address occured when address was negative
      - Fixed bug in clipping version of .q 2-adder routine in MAddToParticles, a3 ↩
        needed to have a skip value added
v1.47 - Fixed bugs in routine PerformGenericBlit2[b], optimised routines were ↩
    copying stencil also but should be data only
      - Added CustomOffsets.l to MScroll family, for using list of custom Width.w, ↩
        X1Offset.w,X2Offset.w,SourceModulo.w values
      - Added CustomOffsets.l to MSScroll family, for using list of custom Width.w ↩
        ,X1Offset.w,X2Offset.w,SourceModulo.w values
      - Fixed bugs in generic blit routines, beq should have been blt for skipping ↩
        if byte loopcounter was empty (-1 not 0)
      - Optimised remainder-byte checking in PerformGenericBlit2[b]/5[b], was no ↩
        need to test for zero bytes
      - Small optimisation, changing bsr to bra if the jump is the last ↩
        instruction of a routine, as there is no need to stack
      - Added LongwordStore.l to data area for temporary storage of longwords, ↩
        like when stacking it would not be possible
      - Fixed bugs in errorchecking routines of MPointShape and MPointCookie, was ↩
        jumping to Error28 instead of Error27
      - Added [,Source.w] parameter to MPoint[Shape][Stencil][Cookie] to specify a ↩
        source bitmap/shape for the operation
      - Added MZoom, MZoomShape, MZoomBitmapToShape and MZoomShapeToBitmap, for ↩
        zoomed unmasked scrolls (+CustomOffsets list!)
      - Added macros CCheckXYFits2 and CCheckXYFits3 for use by errorchecking ↩
        routines of MZoom[Shape][To][Bitmap]
v1.48 - Fixed bug in routine PerformBlit1_16, indirect offsets in SimpleRemap were ↩
    0..3,0..3,0..3,0..3 should have been 0..15
      - Optimised (1fps+) some scrolls that copy the stencil/cookie, as ↩
        interleaving the moves is faster than as bursts
      - Optimised some blit modes in PerformBlit1/2/3[b][_16] using phase-shift, ↩
        pipelining and interleaved mem accesses
      - Optimised the 030 tile routines a bit when doing an STile, by interleaving ↩
        the moves
      - Optimised the SimpleReMap modes in PerformBlit1[b]_16/2[b]_16/3[b]_16 when ↩
        width is multiple of 16 (gains up to 2fps)
      - Optimised zoom routines. Zooms with non-integer x factor are about 4fps ↩
        faster
      - Optimised all zoom routines using addx method instead of swap/swap/add, ↩
        gains of about 3-4fps
      - Added DeRes?.w as last compulsory param of zoom tokens, and added de-res ↩
        routines to zooms
      - Optimised zoom routines for when x factor is 1.0 to do extra-fast y-zoom, ↩
        many fps faster
      - Attempted to rewrite routine PerformLine using .q loop with addx, but it ↩
        performed slower due to need to use divu.w
      - Optimised CustomZoom[DeRes] that uses CustomOffsets.l to do groups of 4 ↩
        pixels if width is multiple, gain up to 2fps+
v1.49 - Added MAddMode which returns mode number 7 for use with various graphics ↩
    routines to choose 'Add' operations
      - Added full support throughout the lib for MAddMode routines, byte values ↩
        of source and dest are simply added together
      - Updated the descriptions of numerous tokens to better represent recent ↩
        changes
```

```
      - Fixed bug in single-colour section of routine PerformReMap, loopcounter ↩
        was screwy
v1.50 - Fixed bugs in MWrapXParticles and MWrapYParticles, loops were BGE and ↩
    should have been BGT, causing 1 illegal loop
      - Fixed bugs in check routines of MBox[F] family, as tables were being ↩
        checked for in MAddMode (where not necessary)
v1.51 - Work resumed 27/02/2000 for the first time since 22/05/1999, copyright ↩
    ownership has been reclaimed
      - Bug identified in PerformBlit1/2/3_16[b] MSimpleReMapMode. Masking isn't ↩
        working right, shape's cookie is blocks of 4!
      - Macro RemakeStencil optimised by changing SNE.b to SEQ.b and removing the ↩
        NOT.l, slight speedup making cookie/stencil
      - Fixed bug in PerformBlit1/2/3_16[b] MSimpleReMapMode. Replaced TST.l with ↩
        CMP.l #-1, as it was skipping 4 instead of 1
      - Design flaw: Programmers should beware that cludged cookies/stencils must ↩
        share the same linemodulo as the main data
      - Optimised macro PerformYFlip for improved speed Yflipping if width is ↩
        multiple of 16
      - Optimised/recoded macro PerformXFlip for improved speed Xflipping if width ↩
         is multiple of 4 or 16. ScrollDemo is 1fps+
```

1.4 mc2pcpumode

```
Mc2pCPUmode CPU.b ; Set cpu c2p uses. Use 'Processor' or MProcessor. <4=030-, ↩
    >3=040+
```

Use this to set the cpu to which the c2p system should be targetted. Use the blitz 'Processor' instruction/token as the parameter, or the new MProcessor command. Values of 0,1,2 and 3 represent the 68030-optimised routine, and 4 or 6 represents the 68040+ optimised routine. You should set the cpu mode for the c2p system before performing a chunky-to-planar conversion, although Mildred will set the default cpu to use according to what cpu is available at runtime. The c2pCPUmode is seperate of the general cpu mode (see later). Mc2pCPUmode affects all c2pWindows that you use because it alters which routine the conversion process uses at all times. It is perfectly safe to use the 040+ c2p routine when the user does not have a processor of that specification, but there may simply be a loss of efficiency.

1.5 mc2pwindow

```
Mc2pWindow c2pWindow#.w,OpWidth.w,OpHeight.w[,SourceBWidth.w[,Processor.b], ↩
    PlanarWidth.w,PlanarHeight.w]
```

Makes a c2pWindow object. This simply describes the size of the chunky-to-planar operation to be later performed. The object does not contain any graphics data and does not convert any data either. A c2pWindow object must be created before a conversion can be performed. Operation width must be a multiple of 32 pixels. If the source and destination 'bitmaps' are a different size, or you have line modulos in some other way, you must supply all the parameters. Specifying processor is done in the same way as Mc2pCPUmode and has precicely the same effect, and is a global setting so needs only be done once.

1.6 mc2pwindowwidth

```
Mc2pWindowWidth [(c2pWindowNumber.w)] ; Returns width of c2pWindow
```

Returns the operation width as defined in a previously created c2pWindow object. If a c2pWindow number is not given, the currently used c2pWindow will be referenced.

1.7 mc2pwindowheight

```
Mc2pWindowHeight [(c2pWindowNumber.w)] ; Returns height of c2pWindow
```

Returns the operation height as defined in a previously created c2pWindow object. If a c2pWindow number is not given, the currently used c2pWindow will be referenced.

1.8 mc2pwindownewheight

```
Mc2pWindowNewHeight c2pWindow#.w,NewHeight.w ; Change height of already defined ↩
    c2p object
```

Once a c2pWindow object has been created you are allowed to alter the operation height any number of times without having to recreate the entire object. Information changed in the c2pWindow structure will be affected by the status of the interlacing features at the time of the object's creation, not the current settings.

1.9 mc2p

```
Mc2p [[c2pWindow#.w],Chunky.l],Planar.l ; Convert chunky to planar (Use Mc2pWindow ↩
     first)
```

Performs a conversion of data from Chunky format to Planar format. Chunky.l points to some chunky-grahics data and Planar.l should point to some planar-graphics data. The planar data, probably held in a normal Blitz Bitmap object or pre-reserved memory area, should be non-interleaved and all bitplanes should be in strictly contigous memory and should not have any line modulos, otherwise the conversion will appear to produce faulty output. Data is read in from chunky (fastram) and output to planar (chipram) with conversion on-the-fly. If you omit the first two parameters (ie omit Chunky.l), the chunky address will be found based on the current chunky Bitmap object's data, and also the Bitmap's handle if Mildred's wrapping feature is active, and also the Bitmap's clip window topleft corner if the Bitmap has clipping active. See the 'Wrap' and 'Clip' related commands. Note that Chunky.l and Planar.l basically represent the top-left corner of the area to be converted and it is with this that you can position the source and destination operations. See also the BitmapPtr and related commands. It is highly recommended that output from the c2p (Planar.l) is aligned to the nearest 4 bytes, ie 32 planar pixels, or the routine will suffer slowdown. Also try and use the highest possible fetchmode in your display. Also note that if the

c2pWindow you use for the operation has no linemodulos, that is that the source and destination are the same width so as to be solid interrupted blocks of memory, then the c2p system will detect this and perform an optimised c2p routine which should be a few fps faster than if you had linemodulos or differing widths.

1.10 mreservec2pwindows

```
MReservec2pWindows [()NumberOfWindows.w[)] ; Reserve structure-memory for  ↩
    c2pWindows
```

This must be called before any operations are performed that have anything to do with c2pWindow objects, unless making use of the default reserved c2pWindow objects. This command simply reserves some structure memory in order to hold the objects, but does not make space for holding any graphics-data. c2pWindow objects are numbered from 0 upwards. If used as a function, this command will return the address of where the c2pWindow objects are being stored, or 0 for failure.

1.11 mreserveshapes

```
MReserveShapes [()NumberOfShapes.w[,ShapeBankToUse.w][)] ; Reserve structure- ↩
    memory for Shapes
```

This must be called before any operations are performed that have anything to do with chunky Shape objects, unless making use of the default reserved chunky Shapes. A chunky Shape is one of Mildred's 'shape' objects, akin to a normal blitz shapes except that they hold different information and the graphics are stored in a different format (chunky). Using this command does not allocate space for any shape graphics, but does make space for the shapes to be created and stored. Chunky Shapes are numbered from 0 upwards. If used as a function, this command will return the address of where the chunky Shape objects (structures) are being stored, or 0 for failure. If the optional ShapeBankToUse parameter is specified, with a valid shape bank number in the range 0..31, a particular bank will be 'used' before new space for chunky Shapes is allocated, meaning that you do not have to perform a seperate call to MUseShapeBank.

1.12 mreservebitmaps

```
MReserveBitmaps [()NumberOfBitmaps.w[)] ; Reserve structure-memory for Bitmaps
```

This must be called before any operations are performed that have anything to do with chunky Bitmap objects, unless making use of the default reserved chunky Bitmaps. A chunky Bitmap is one of Mildred's 'bitmap' objects, akin to a normal blitz bitmap except that they hold different information and the graphics are stored in a different format (chunky). Using this command does not allocate space for any shape graphics, but does make space for the shapes to be created and stored. Chunky Shapes are numbered from 0 upwards. If used as a function, this command will return the address of where the chunky Bitmap objects (structures) are being stored, or 0 for failure.

1.13 mshape

```
MShape [()ShapeNumber.w,Width.w,Height.w[)] ; Allocmem for shape data
```

Initialises a new shape, which basically means that it creates a new chunky Shape object. If Mildred's autocookie feature is turned on, a cookie for the shape will also be initialised also. Width should be a multiple of 4. A chunky Shape object contains graphics data (also cookie-data). If used as a function, this command will return the address of the new Shape's graphics data or 0 for failure.

1.14 mbitmap

```
MBitmap [()BitmapNumber.w,Width.w,Height.w[)] ; Allocmem for bitmap data
```

Initialises a new bitmap, which basically means that is creates a new chunky Bitmap object. If Mildred's autostencil feature is turned on, a stencil for the bitmap will also be initialised. Width should be a multiple of 4 but bear in mind that if you use any of the 'Block' or 'Tile' commands on 040+ it is necessary to have a Bitmap width in multiples of 16. A chunky Bitmap object contains graphics data (also stencil-data). Unlike normal blitz bitmap's, a chunky Bitmap in Mildred has stencils as integral parts of the object in the same way that shapes have integral cookies. If used as a function, this command will return the address of the new Bitmap's graphics data or 0 for failure.

1.15 mautocookie

```
MAutoCookie On/Off ; Autocreation of ByteForByte cookies
```

Switches the auto-cookie-creation feature of Mildred On or Off. If switched On, any chunky Shapes created in future will automatically create a cookie also.

1.16 mautostencil

```
MAutoStencil On/Off ; Autocreation of ByteForByte stencils
```

Switched the auto-stencil-creation feature of Mildred On or Off. If switched On, any chunky Bitmaps created in future will automatically create a stencil also (similar to a cookie).

1.17 mfreec2pwindow

```
MFreec2pWindow [Firstc2pWindow.w[,Lastc2pWindow.w]] ; Free a c2pWindow, range of ↩
    c2pWindows or all c2pWindows
```

Allows you to free a specific, all, or a range of c2pWindow objects. As c2pWindow objects do not contain any graphics-data, this will simply kill the object definition. If you omit the parameters it will free all objects regardless.

If you specify a second parameter it will free a range of objects (and will error-check that they exist first).

1.18 mfreeshape

```
MFreeShape [FirstShape.w[,LastShape.w]] ; Free a Shape, range of shapes, or all ↩
    shapes
```

Allows you to free a specific, all, or a range of chunky Shape objects. Chunky Shape objects contain graphics and possibly cookie data, so if you free these objects the data will be freed also. If you omit the parameters it will free all objects unconditionally. If you specify a second parameter it will check that each one exists first and if not will generate an error. Specifying a second parameter will free a range of shapes.

1.19 mfreebitmap

```
MFreeBitmap [FirstBitmap.w[,LastBitmap.w]] ; Free a Bitmap, range of bitmaps, or ↩
    all bitmaps
```

Allows you to free a specific, all, or a range of chunky Bitmap objects. Chunky Bitmap objects contain graphics and possibly stencil data, so if you free these objects the data will be freed also. If you omit the parameters it will free all objects unconditionally. If you specify a range of objects to free it will check that each one exists first and if not will generate an error.

1.20 mshapewidth

```
MShapeWidth [(ShapeNumber.w)] ; Returns width of Shape
```

Returns the width of a previously created chunky Shape object. If a shape number is not specified the currently used shape will be referenced.

1.21 mbitmapwidth

```
MBitmapWidth [(BitmapNumber.w)] ; Returns width of Bitmap
```

Returns the width of a previously created chunky Bitmap object. If a bitmap number is not specified the currently used bitmap will be referenced.

1.22 mshapeheight

```
MShapeHeight [(ShapeNumber.w)] ; Returns height of Shape
```

Returns the height of a previously created chunky Shape object. If a shape number is not specified the currently used shape will be referenced.

1.23 mbitmapheight

```
MBitmapHeight [(BitmapNumber.w)] ; Returns height of Bitmap
```

Returns the height of a previously created chunky Bitmap object. If a bitmap number is not specified the currently used bitmap will be referenced.

1.24 maddrc2pwindow

```
MAddrc2pWindow [(c2pWindowNumer.w)] ; Returns address of c2pWindow structure
```

Returns the address in memory of a specified c2pWindow object. This is the address at which the object's structure can be found. If a c2pWindow number is not specified then the currently used c2pWindow will be referenced.

1.25 maddrshape

```
MAddrShape [(ShapeNumber.w)] ; Returns address of Shape structure
```

Returns the address in memory of a specified chunky Shape object. This is the address at which the object's structure can be found, not the address of the graphics-data or cookie-data. If a shape number is not specified then the currently used shape will be referenced.

1.26 maddrbitmap

```
MAddrBitmap [(BitmapNumber.w)] ; Returns address of Bitmap structure
```

Returns the address in memory of a specified chunky Bitmap object. This is the address at whicih the object's structure can be found, not the address of the graphics-data or stencil-data. If a bitmap number is not specified then the currently used bitmap will be referenced.

1.27 mhandle

```
MHandle ShapeNumber.w,XOffset.w,YOffset.w ; Set handle of Shape
```

Sets the handle for a previously-created chunky Shape object. This is the coordinate offset that is added to the coordinates that you may specify for a blit-operation later on. These coordinates are also used by Mildred's wrapping feature to provide the offset of a sliding window relative to the base of the Shape's image and/or cookie.

1.28 mbitmaporigin

```
MBitmapOrigin BitmapNumber.w,XOffset.w,YOffset.w ; Set origin of Bitmap
```

Sets the handle for a previously-created chunky Bitmap object. This is the coordinate offset that is added to the coordintes that you may specify for a blit-operation later on, but is more likely to be used in conjunction with Mildred's wrapping feature to provide the offset of a sliding window relative to the Bitmap's image and/or stencil.

1.29 musec2pwindow

```
MUsec2pWindow Mainc2pWindowNum.w[,Secondc2pWindowNum.w[,Thirdc2pWindowNum.w]] ;  ↩
    Current to use
```

Allows you to specify one, two or three c2pWindow objects to be used as the 'current' objects. Once made current, an object can be 'assumed' in later commands so that it doesn't have to be respecified each time.

1.30 museshape

```
MUseShape MainShapeNum.w[,SecondShapeNum.w[,ThirdShapeNum.w]] ; Current Shape(s)  ↩
    to use
```

Allows you to specify one, two or three chunky Shape objects to be used as the 'current' objects. Once made current, an object can be 'assumed' in later commands so that it doesn't have to be respecified each time.

1.31 musebitmap

```
MUseBitmap MainBitmapNum.w[,SecondBitmapNum.w[,ThirdBitmapNum.w]] ; Current Bitmap ↩
     to use
```

Allows you to specify one, two or three chunky Bitmap objects to be used as the 'current' objects. Once made current, an object can be 'assumed' in later commands so that it doesn't have to be respecified each time.

1.32 musedc2pwindow

```
MUsedc2pWindow  ; Returns currently used c2pWindow
```

Returns the number of the currently used main c2pWindow object.

1.33 musedshape

```
MUsedShape  ; Returns currently used Shape
```

Returns the number of the currently used main chunky Shape object.

1.34 musedbitmap

```
MUsedBitmap  ; Returns currently used Bitmap
```

Returns the number of the currently used main chunky Bitmap object.

1.35 mcludgeshape

```
MCludgeShape ShapeNumber.w,Width.w,Height.w,Memory.l ; Cludge shape from existing ↩
    mem
```

Creates a new chunky Shape object from some existing graphics memory that may or may not contain some chunky-format graphics. Width should be a multiple of 4. The Memory.l base address should be aligned to a multiple of 16 bytes in memory. If the autocookie feature of Mildred is active some new memory will also be allocated for a cookie (not cludged). The memory you cludge a Shape onto should generally be in fastram.

1.36 mcludgebitmap

```
MCludgeBitmap BitmapNumber.w,Width.w,Height.w,Memory.l ; Cludge bitmap from  ↩
    existing mem
```

Creates a new chunky Bitmap object from some existing graphics memory that may or may not contain some chunky-format graphics. Width should be a multiple of 4. The Memory.l base address should be aligned to a multiple of 16 bytes in memory. If the autostencil feature of Mildred is active some new memory will also be allocated for a stencil (not cludged). The memory you cludge a Bitmap onto should generally be in fastram.

1.37 mautousec2pwindows

```
MAutoUsec2pWindows True/False ; Automatically 'use' new c2pWindows. <>0=True
```

If switched on, it causes all c2pWindow objects that are created from this point onwards to be 'used' as the 'current' main c2pWindow object.

1.38 mautouseshapes

```
MAutoUseShapes True/False ; Automatically 'use' new shapes. <>0=True
```

If switched on, it causes all chunky Shape objects that are created from this point onwards to be 'used' as the 'current' main chunky Shape object.

1.39 mautousebitmaps

```
MAutoUseBitmaps True/False ; Automatically 'use' new bitmaps. <>0=True
```

If switched on, it causes all chunky Bitmap objects that are created from this point onwards to be 'used' as the 'current' main chunky Bitmap object.

1.40 mmakecookie

```
MMakeCookie [FirstShape.w[,LastShape.w]] ; Make a cookie for a shape, range of ↩
    shapes, or all shapes
```

Initialises (if necessary) and generates cookies for a specific shape, a range of shapes, or all shapes. If the cookie doesn't yet exist it will be allocated. The cookie will be generated based on the contents of the Shape's image. If the parameters are omitted, all currently created chunky Shapes will have cookies generated. The area of a Shape used to make a cookie can be cropped using the Shape's clip window. If a second parameter is specified you can make cookies for a range of shapes.

1.41 mmakestencil

```
MMakeStencil [FirstBitmap.w[,LastBitmap.w]] ; Make a stencil for a bitmap, range ↩
    of bitmaps, or all bitmaps
```

Initialises (if necessary) and generates stencils for a specific bitmap, a range of bitmaps, or all bitmaps. If the stencil doesn't yet exist it will be allocated. The stencil will be generated based on the contents of the Bitmap's image. If the parameters are omitted, all currently created chunky Bitmaps will have stencils generated. The area of a Bitmap used to make a stencil can be cropped using the Bitmap's clip window. If a second parameter is specified you can make stencils for a range of bitmaps.

1.42 mfreecookie

```
MFreeCookie [FirstShape.w[,LastShape.w]] ; Free a Shape's cookie, a range of Shape ↩
    's cookies, or all cookies
```

Free's a speific, all or a range of previously-created cookies in previously-created chunky Shape objects. Only the cookie is freed. If the

parameters are omitted all cookies in any created chunky Shapes will be freed. If a second parameter is specified then a range of shape's cookies will be freed.

1.43 mfreestencil

```
MFreeStencil [FirstBitmap.w[,LastBitmap.w]] ; Free's a Bitmap's stencil, a range ↩
    of stencils or all stencils
```

Free's a specific, all or a range of previously-created stencils in previously-created chunky Bitmap objects. Only the stencil is freed. If the parameters are omitted all stencils in any created chunky Bitmaps will be freed. If you specify the second parameter then a range of bitmap's stencils will be freed.

1.44 mautoshapewrap

```
MAutoShapeWrap On/Off ; Auto X&Y Handle-wrapping for Shapes
```

Turns the autowrapping feature of Mildred for chunky Shapes On or Off. Generally, if the Bitmap is the destination in a graphics operation, the Shape's handle will be added to the output coordinates (and applies at all times even when output coordinates are not specified). When wrapping is on, the base address will have an offset added to it to represent the handle coordinates, before the graphics-operation is performed. This causes the graphics-operation to output into possibly unreserved memory and Mildred does not check for this so be very careful to allocate a memory area to move into (make the Shape big enough). This command turns the automatic wrapping On or Off which means that when a new chunky Shape is created, the 'wrapping' flag for the Shape is activated to indicate that the handle will be used in the operation. Wrapping has been implemented mainly to better support sliding windows in memory for scrolling.

1.45 mautobitmapwrap

```
MAutoBitmapWrap On/Off ; Auto X&Y Handle-Wrapingp for Bitmaps
```

Turns the autowrapping feature of Mildred for chunky Bitmaps On or Off. Generally, if the Bitmap is the destination in a graphics operation, the Bitmap's handle will be added to the output coordinates (and applies at all times even when output coordinates are not specified). When wrapping is on, the base address will have an offset added to it to represent the handle coordinates, before the graphics-operation is performed. This causes the graphics-operation to output into possibly unreserved memory and Mildred does not check for this so be very careful to allocate a memory area to move into (make the Bitmap big enough). This command turns the automatic wrapping On or Off which means that when a new chunky Bitmap is created, the 'wrapping' flag for the Bitmap is activated to indicate ↩
    that
the handle will be used in the operation. Wrapping has been implemented mainly to better support sliding windows in memory for scrolling.

1.46 mshapewrap

```
MShapeWrap ShapeNumber.w,On/Off ; De/Activate X&Y Handle-Wrap for Shape
```

Specifically turns the wrapping for a chunky Shape On or Off. When On, the Shape's handle may be used in creating the address of part of a graphics operation, typically when outputting to the Shape. This is mainly used for scroll methods. Wrapping is used in probably all graphics operations in Mildred.

1.47 mbitmapwrap

```
MBitmapWrap BitmapNumber.w,On/Off ; De/Activate X&Y Handle-Wrap for Bitmap
```

Specifically turns the wrapping for a chunky Bitmap On or Off. When On, the Bitmap's handle may be used in creating the address of part of a graphics operation, typically when outputting to the Bitmap. This is mainly used for scroll methods. Wrapping is used in probably all graphics operations in Mildred.

1.48 mcludgeshapestruct

```
MCludgeShapeStruct [()SourceShape.w,DestShape.w[)] ; Copy definition-data only
```

Creates a new chunky Shape object based on an existing Shape object. The existing Shape's definition is duplicated into the new chunky Shape and the graphics and cookie data are cludged to use the data that exists in the original chunky Shape. Graphics and cookie data are not copied to the new chunky Shape. If used as a function, this command will return the address of the new Shape's graphic data, or 0 for failure.

1.49 mcludgebitmapstruct

```
MCludgeBitmapStruct [()SourceBitmap.w,DestBitmap.w[)] ; Copy definition-data only
```

Creates a new chunky Bitmap object based on an existing Bitmap object. The existing Bitmap's definition is duplicated into the new chunky Bitmap and the graphics and stencil data are cludged to use the data that exists in the original chunky Bitmap. Graphics and stencil data are not copied to the new chunky Bitmap. If used as a function, this command will return the address of the new Bitmap's graphic data, or 0 for failure.

1.50 mcopyc2pwindow

```
MCopyc2pWindow Sourcec2pWindow.w,Destc2pWindow.w ; Copy definition-data only
```

Duplicates an existing c2pWindow object into a new one. As c2pWindows do not contain any graphics data, the c2pWindow's definition data describing the size of the operation only is copied, so is effectively just a 'clone' of the original. You may want to use this in conjunction with MNewc2pWindowHeight.

1.51 mshapewindow

```
MShapeWindow [()SourceShape.w,DestShape.w,X.w,Y.w,Width.w,Height.w[)] ; Cludge ↩
    Shape within a Shape
```

Makes a new chunky Shape object based on an existing chunky Shape object, allowing a 'window' within the existing Shape to be used as the new one. Therefore graphics and cookie data are cludged from the existing Shape's data. Width should be a multiple of 4 and X,Y specify the top-left corner of the window within the existing chunky Shape. If used as a function, this command will return the address of the Shape's graphics data, or 0 for failure.

1.52 mbitmapwindow

```
MBitmapWindow [()SourceBitmap.w,DestBitmap.w,X.w,Y.w,Width.w,Height.w[)] ; Cludge ↩
    Bitmap within a Bitmap
```

Makes a new chunky Bitmap object based on an existing chunky Bitmap object, allowing a 'window' within the exising Bitmap to be used as the new one. Therefore graphics and sencil data are cludged from the existing Bitmap's data. Width should be a multiple of 4 and X,Y specify the top-left corner of the window within the existing chunky Bitmap. If used as a function, this command will return the address of the Bitmap's graphics data, or 0 for failure.

1.53 mbitmapshape

```
MBitmapShape [()SourceBitmap.w,DestShape.w[)] ; Copy definition-data only
```

Creates a new chunky Shape object based on a previously-created chunky Bitmap object. The information for the Bitmap is copied into the Shape and then the graphics are cludged. The Bitmap's stencil will be used as the new Shape's cookie. The Bitmap's graphic and stencil are not copied. This command takes advantage of the fact that, internally, chunky Shapes are 100% interchanageable with chunky Bitmaps. If used as a function, this command will return the address of the Shape's graphics data, or 0 for failure.

1.54 mshapesbitmap

```
MShapesBitmap [()SourceShape.w,DestBitmap.w[)] ; Copy definition-data only
```

Creates a new chunky Bitmap object based on a previously-created chunky Shape object. The information for the Shape is copied into the Bitmap and then the graphics are cludged. The Shape's cookie will be used as the new Bitmap's stencil. The Shape's graphic and cookie are not copied. This command takes advantage of the fact that, internally, chunky Bitmaps are 100% interchangeable with chunky Shapes. If used as a function, this command will return the address of the Bitmap's graphics data, or 0 for failure.

1.55 mautocookiexflip

```
MAutoCookieXFlip On/Off ; Auto X-Flip for Shape's cookie
```

Sets Mildred's autocookieXFlip setting On or Off. If On, any X (horizontal) flip operations performed on a chunky Shape will also be performed on the Shape's cookie.

1.56 mautocookieyflip

```
MAutoCookieYFlip On/Off ; Auto Y-Flip for Shape's cookie
```

Sets Mildred's autocookieYFlip setting On or Off. If On, any Y (verticle) flip operations performed on a chunky Shape will also be performed on the Shape's cookie.

1.57 mautostencilxflip

```
MAutoStencilXFlip On/Off ; Auto X-Flip for Bitmap's stencil
```

Sets Mildred's autostencilXFlip setting On or Off. If On, any X (horizontal) flip operations performed on a chunky Bitmap will also be performed on the Bitmap's stencil.

1.58 mautostencilyflip

```
MAutoStencilYFlip On/Off ; Auto Y-Flip for Bitmap's stencil
```

Sets Mildred's autostencilYFlip setting On or Off. If On, any Y (verticle) flip operations performed on a chunky Bitmap will also be performed on the Bitmap's stencil.

1.59 mautocookieflip

```
MAutoCookieFlip On/Off ; Auto X&Y Cookie-Flip for Shapes
```

Sets Mildred's autocookieYFlip and autocookieXFlip On or Off. If On, any X (horizontal) or Y (verticle) flip operations performed on a chunky Shape will also be performed on the Shape's cookie. This command overrides any previous seperate calls to MAutoCookieXFlip and MAutoCookieYFlip.

1.60 mautostencilflip

```
MAutoStencilFlip On/Off ; Auto X&Y Stencil-Flip for Bitmaps
```

Sets Mildreds autostencilYFlip and autostencilXFlip On or Off. If On, any X (horizontal) or Y (verticle) flip operations performed on a chunky Bitmap will also be performed on the Bitmap's stencil. This command overrides any previous seperate calls to MAutoStencilXFlip and MAutoStencilYFlip.

1.61 mshapexflip

```
MShapeXFlip ShapeNumber.w ; Horizontally flip a Shape (see MAutoCookieFlip)
```

Flips a previously-created chunky Shape's graphic horizontally. The operation may also be performed on the cookie if MAutoCookieXFlip or MAutoCookieFlip have been activated. The area of the Shape to be flipped can be cropped using the Shape's clip window.

1.62 mshapeyflip

```
MShapeYFlip ShapeNumber.w ; Vertically flip a Shape (see MAutoCookieFlip)
```

Flips a previously-created chunky Shape's graphic vertically. The operation may also be performed on the cookie if MAutoCookieYFlip or MAutoCookieFlip have been activated. The area of the Shape to be flipped can be cropped using the Shape's clip window.

1.63 mbitmapxflip

```
MBitmapXFlip BitmapNumber.w ; Horizontally flip a Bitmap (see MAutoStencilFlip)
```

Flips a previously-created chunky Bitmap's graphic horizontally. The operation may also be performed on the stencil if MAutoStencilXFlip or MAutoStencilFlip have been activated. The area of the Bitmap to be flipped can be cropped using the Shape's clip window.

1.64 mbitmapyflip

```
MBitmapYFlip BitmapNumber.w ; Vertically flip a Bitmap (see MAutoStencilFlip)
```

Flips a previously-created chunky Bitmap's graphic vertically. The operation may also be performed on the stencil if MAutoStencilYFlip or MAutoStencilFlip have ben activated. The area of the Bitmap to be flipped can be cropped using the Bitmap's clip window.

1.65 mcookiexflip

```
MCookieXFlip ShapeNumber.w ; Horizontally flip a Shape's cookie
```

Flips the cookie of a previously-created chunky Shape object horizontally. This does not affect the Shape's image. The area of the cookie to be flipped can be cropped using the Shape's clip window.

1.66 mcookieyflip

```
MCookieYFlip ShapeNumber.w ; Vertically flip a Shape's cookie
```

Flips the cookie of a previously-created chunky Shape object vertically. This does not affect the Shape's image. The area of the cookie to be flipped can be cropped using the Shape's clip window.

1.67 mstencilxflip

```
MStencilXFlip BitmapNumber.w ; Horizontally flip a Bitmap's stencil
```

Flips the stencil of a previously-created chunky Bitmap object horizontally. This does not affect the Bitmap's image. The area of the stencil to be flipped can be cropped using the Bitmap's clip window.

1.68 mstencilyflip

```
MStencilYFlip BitmapNumber.w ; Vertically flip a Bitmap's stencil
```

Flips the stencil of a previously-created chunky Bitmap object vertically. This does not affect the Bitmap's image. The area of the stencil to be flipped can be cropped using the Bitmap's clip window.

1.69 mautoshapeclip

```
MAutoShapeClip Status.b ; Auto-clip new Shapes. On/Off
```

Sets the clipping flag of a chunky Shape On or Off. If On, some graphics operations performed on the Shape may be restrained to affect only the area of the Shape defined by the clip window.

1.70 mautobitmapclip

```
MAutoBitmapClip Status.b ; Auto-clip new Bitmaps. On/Off
```

Sets the clipping flag of a chunky Bitmap On or Off. If On, some graphics operations performed on the Bitmap may be restrained to affect only the area of the Bitmap defined by the clip window.

1.71 mshapeclip

```
MShapeClip ShapeNumber.w[,X.w,Y.w,Width.w,Height.w][,Active] ; Active=On/Off.  ↩
    Define Shape's clip window
```

Specifies the position (X,Y) and size (WidthxHeight) of the clip window to be used for the specified chunky Shape object. Clipping may also be turned On or Off using the additional 'active' parameter. The window should be fully within the limits of the Shape's graphic and the Width should be a multiple of 4.

1.72 mbitmapclip

```
MBitmapClip BitmapNumber.w[,X.w,Y.w,Width.w,Height.w][,Active] ; Active=On/Off.  ↩
    Define Bitmap's clip window
```

Specifies the position (X,Y) and size (WidthxHeight) of the clip window to be used for the specified chunky Bitmap object. Clipping may also be turned On or Off using the additional 'active' parameter. The window should be fully within the limits of the Bitmap's graphic and the Width should be a multiple of 4.

1.73 mgetashape

```
MGetaShape ShapeNumber.w,X.w,Y.w,Width.w,Height.w[,Block?,StencilIsCookie?] ; Grab ↩
     shape from bitmap
```

Grabs a new chunky Shape object from a previously-created chunky Bitmap object. The chunky Shape is created of the dimensions specified (Width must be multiple of 4) and then the Bitmap's graphic is copied into the Shape. If the 'Block' parameter is set to True or On, a block-scroll type of grab will be used, for which the Shape should have a width that is a multiple of 16, and the Shape should be grabbed from X coordinates that are multiples of 16. You can also specify to use the Bitmap's stencil as the cookie, in which case the relevant portion of the stencil will be copied into the new Shape's cookie. The end result is a new, standalone (non-cludged) chunky Shape.

1.74 mgetabitmap

```
MGetaBitmap BitmapNumber.w,X.w,Y.w,Width.w,Height.w[,Block?,CookieIsStencil?] ;  ↩
    Grab bitmap from shape
```

Grabs a new chunky Bitmap object from a previously-created chunky Shape object.

The chunky Bitmap is created of the dimensions specified (Width must be multiple of 4) and then the Shape's graphic is copied into the Bitmap. If the 'Block' parameter is set to True or On, a block-scroll type of grab will be used, for which the Bitmap should have a width that is a multiple of 16, and the Bitmap should be grabbed from X coordinates that are multiples of 16. You can also specify to use the Shape's cookie as the stencil, in which case the relevant portion of the cookie will be copied into the new Bitmap's stencil. The end result is a new, standalone (non-cludged) chunky Bitmap.

1.75 mscroll

```
MScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[,CustomOffsets.l]] ↩
    ; Copy graphic
```

Copies an area from within one chunky Bitmap to somewhere in another chunky Bitmap. The currently used main Bitmap is the destination, and is also the source unless you specify otherwise. The entire operation should fit within the dimensions of the chunky Bitmaps and will not be cropped. If CustomOffsets.l is specified then a list of custom settings will be used for each row of the scroll. CustomOffsets.l should point to an array of data which should contain sets of Width.w,X1Offset.w,X2Offset.w,SourceModulo.w. Each group of 4 variables correspond to a single line in the scroll and there should be enough data for all of the lines. Width.w is an absolute width, which should be at least 1, and is the width of the operation for a given line. X1Offset.w is a relative offset in the source X1 coordinate, relative to the previous line. For example, to maintain the same X1 coordinate throughout the scroll you should have X1Offset.w set to 0 for every line. Setting it to 1 for every line would cause the scroll to slant at 45 degrees, for example. X2Offset.w corresponds to the destination X2 coordinate and is relative to the X2 position of the previous line. An X2Offset of 0 on every line produces a normal rectangle. Note that X1Offset and X2Offset take effect prior to the start of the processing of a given line. On the contrary, SourceModulo.w takes effect when a given line has finished processing. It is an absolute value which is added to the source position at the end of a line to effect a 'modulo' offset. The modulo for each line is relative to the previous line so setting SourceModulo.w to 0 for every line will produce a normal blit. It is possible to specify a negative modulo, equal to the width of the bitmap, such as -320, in order to repeat the first line of the source over and over again. Similarly, -320*2 (-640) would cause an on-the-fly verticle flip of the area, for which you should logically point the source Y1 coordinate to the bottom of the area rather than the top. Used carefully the SourceModulo will allow for variable verticle zooming. Note that if you specify the source and destination bitmaps as being the same bitmap there is a small chance that an undesired effect will be produced.

1.76 mscrollshape

```
MScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[,CustomOffsets ↩
    .l]] ; Copy graphic
```

Does the same as MScroll but with regards to chunky Shape objects. Source and destination are chunky Shape objects.

1.77 mscrollstencil

```
MScrollStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
    CustomOffsets.l]] ; Copy sten to sten
```

Does the same as MScroll but with regards to the stencil of chunky Bitmap objects. Source and destination are chunky Bitmap object's stencils. The source Bitmap's stencil will be copied into the destination Bitmap's stencil.

1.78 mscrollcookie

```
MScrollCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]] ;Copy cook to cook
```

Does the same as MScroll but with regards to the cookie of chunky Shape objects. Source and destination are chunky Shape object's cookies. The source Shape's cookie will be copied into the destination Shape's cookie.

1.79 mmaskscroll

```
MMaskScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Copy bitmap ↩
     graphic with stencil-cut
```

Performs a masked copy of an area from within one chunky Bitmap to somewhere in another chunky Bitmap. The currently used main Bitmap is the destination, and is also the source unless you specify otherwise. Width should be a multiple of 4 and the entire operation should fit within the dimensions of the chunky Bitmaps and will not be cropped. Any chunky Bitmaps used should have a stencil created for them and this will be used to perform a masked copy. Any holes in the stencil, ie any areas of colour 0 in the source Bitmap, will be transparent.

1.80 mmaskscrollshape

```
MMaskScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy  ↩
    shape graphic with cookie-cut
```

Does the same as MMaskScroll but with regards to chunky Shape objects. Source and destination are chunky Shape objects.

1.81 mmaskscrollstencil

```
MMaskScrollStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w];Copy  ↩
    stencil2stencil & stencil-cut
```

Does the same as MMaskScroll but with regards to the stencil of chunky Bitmap objects. Source and destination are chunky Bitmap object's stencils. The source Bitmap's stencil will be mask-blitted onto the destination Bitmap's stencil.

1.82 mmaskscrollcookie

```
MMaskScrollCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy ↩
    cookie to cookie & cookie-cut
```

Does the same as MMaskScroll but with regards to the cookie of chunky Shape objects. Source and destination are chunky Shape object's cookie. The source Shape's cookie will be mask-blitted onto the destination Shape's cookie.

1.83 mscrollbitmaptoshape

```
MScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
    CustomOffsets.l]];bitmap 2 shape
```

Does the same as MScroll except that the source of the operation will be a chunky Bitmap and the destination will be a chunky Shape. The Bitmap's image will be copied into the Shape.

1.84 mscrollshapetobitmap

```
MScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]] ;shape 2 bitmap
```

Does the same as MScroll except that the source of the operation will be a chunky Shape and the destination will be a chunky Bitmap. The Shape's image will be copied into the Bitmap. This does effectively the same as a normal 'Blit' of a Shape to a Bitmap.

1.85 mscrollstenciltocookie

```
MScrollStencilToCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
    CustomOffsets.l]]; sten2cookie
```

Does the same as MScroll except that the source of the operation will be the stencil of a chunky Bitmap, which will be copied to the cookie of a chunky Shape.

1.86 mscrollcookietostencil

```
MScrollCookieToStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]] ; cookie2sten
```

Does the same as MScroll except that the source of the operation will be the cookie of a chunky Shape, which will be copied to the stencil of a chunky Bitmap.

1.87 mmaskscrollbitmaptoshape

```
MMaskScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ↩
    ; Copy bitmap to shape & cut
```

Does the same as MMaskScroll except that the source of the operation will be a chunky Bitmap and the destination will be a chunky Shape. The Bitmap's image will be blitted to the shape using the Bitmap's stencil as a mask.

1.88 mmaskscrollshapetobitmap

```
MMaskScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; ↩
     Copy shape to bitmap & cut
```

Does the same as MMaskScroll except that the source of the operation will be a chunky Shape and the destination will be a chunky Bitmap. The Shape's image will be blitted to the Bitmap using the Shape's cookie as a mask. This does effictively the same as a normal 'Cookie Blit' of a Shape to a Bitmap.

1.89 mmaskscrollstenciltocookie

```
MMaskScrollStencilToCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w ↩
    ] ;Copy stencil2cookie & cut
```

Does the same as MMaskScroll except that the source of the operation will be a chunky Bitmap's stencil which will be mask-blitted to the cookie of a chunky Shape.

1.90 mmaskscrollcookietostencil

```
MMaskScrollCookieToStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ↩
     ; Copy cookie2stencil & cut
```

Does the same as MMaskScroll except that the source of the operation will be a chunky Shape's cookie which will be mask-blitted to the stencil of a chunky Bitmap.

1.91 mblockscroll

```
MBlockScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; BlockCopy ↩
    graphic
```

Performs a block-scroll within a chunky Bitmap object or using a different chunky Bitmap as the source. Width should be a multiple of 16 and the overall operation should be within the dimensions of the Bitmaps. X1 (source) and X2 (dest) should be a multiple of 16 as a special block-copy (in multiples of 16) is performed.

1.92 mblockscrollshape

```
MBlockScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    BlockCopy graphic
```

Does the same as MBlockScroll except with regards to chunky Shapes. The Shape's image will be copied into another Shape's image. Width, X1 and X2 should be multiples of 16.

1.93 mblockscrollstencil

```
MBlockScrollStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;  ↩
    BlockCopy stencil to stencil
```

Does the same as MBlockScroll except with regards to the stencils of chunky Bitmaps. The Bitmap's stencil will be copied into another Bitmap's stencil. Width, X1 and X2 should be multiples of 16.

1.94 mblockscrollcookie

```
MBlockScrollCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    BlockCopy cookie to cookie
```

Does the same as MBlockScroll except with regards to the cookies of chunky Shapes. The Shape's cookie will be copied into another Shape's cookie. Width, X1 and X2 should be multiples of 16.

1.95 mblockscrollbitmaptoshape

```
MBlockScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ↩
    ; BlockCopy bitmap to shape
```

Does the same as MBlockScroll except that the source of the operation is a chunky Bitmap and the destination is a chunky Shape.

1.96 mblockscrollshapetobitmap

```
MBlockScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w]  ↩
    ; BlockCopy shape to bitmap
```

Does the same as MBlockScroll except that the source of the operation is a chunky Shape and the destination is a chunky Bitmap. This is effectively a 'Block' blit of a Shape to a Bitmap.

1.97 mblockscrollstenciltocookie

```
MBlockScrollStencilToCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum. ↩
    w];BlockCopy stencil2cookie
```

Does the same as MBlockScroll except that the source of the operation is a chunky Bitmap's stencil which is block-copied into the cookie of a chunky Shape.

1.98 mblockscrollcookietostencil

```
MBlockScrollCookieToStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w ↩
    ] ; BlockCopy cookie2stencil
```

Does the same as MBlockScroll except that the source of the operation is a chunky Shape's cookie which is block-copied into the stencil of a chunky Bitmap.

1.99 mcpu

```
MCPU Processor.b ; Set cpu routines allowed to use. 0..3=000..030, or >3=040+.  ↩
    CAREFUL!!
```

Sets the general CPU specification that the user has available. This will enable Mildred to use specially optimised routines where possible to provide better performance on particular processors. You should pass the blitz 'Processor' instruction or the new MProcessor command as the parameter in which values of 0..3 represent that the user has a 68030 or lower, and a 4 or 6 represents that they have a 68040 or higher. Be sure that the user does have the processor you are specifying as, unlike Mc2pCPUmode (which is seperate entirely), it is possible to crash the computer if you attempt to use routines for a cpu that is not capable of certain instructions (such as 040+ routines). This problem is usually avoided by simply using the blitz 'Processor' instruction or MProcessor. MCPU should be used near to the start of your program before performing any operations that use chunky Shapes and chunky Bitmaps, although since v1.36 it will be automatically set internally according to what cpu is available at runtime. If you do not wish to override this you can forget about having to set a CPU mode.

1.100 mcls

```
MCls [Colour] Clear a bitmap to colour 0 or the specified colour
```

Performs a clearscreen on the currently used main chunky Bitmap. If specified, it will be cleared to the colour you choose, otherwise will be cleared to 0 (usually black). The area of the Bitmap affected by the operation can be limited by use of a clip window and the clipping feature in the Bitmap to be used. See MDrawingMode for a decription of the drawing modes that are available for MCls.

1.101 mclsshape

MClsShape [Colour] Clear a shape to colour 0 or the specified colour

Does the same as MCls but on a Shape's graphic and using the currently used main chunky Shape.

1.102 mclsstencil

MClsStencil [Colour] Clear a stencil to colour 0 or the specified colour

Does the same as MCls but on the stencil of the currently used chunky Bitmap. The actual value placed into the stencil /represents/ the colour you specify as if all pixels in the graphic were of that colour. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the stencil.

1.103 mclscookie

MClsCookie [Colour] Clear a cookie to colour 0 or the specified colour

Does the same as MCls but on the cookie of the currently used chunky Shape. The actual value placed into the cookie /represents/ the colour you specify as if all pixels in the graphic were of that colour. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the cookie.

1.104 mplot

MPlot Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the bitmap [to the specified ↩ colour]

Plots a single pixel within the currently used main Chunky Bitmap to the specified colour or the currently used ink colour. See MDrawingMode for further details about the possible drawing modes available.

1.105 mplotshape

MPlotShape Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the shape [to the ↩ specified colour]

Plots a single pixel within the currently used main chunky Shape to the specified colour or the currently used ink colour.

1.106 mplotstencil

```
MPlotStencil Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the stencil to * ↩
    represent* the [specified] colour
```

Plots a single pixel within the stencil of the currently used main chunky Bitmap to /represent/ the specified colour or the currently used ink colour in the Bitmap's graphic. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the stencil.

1.107 mplotcookie

```
MPlotCookie Xpos.w,Ypos.w[,Colour] ; Plot a single pixel in the cookie to * ↩
    represent* the [specified] colour
```

Plots a single pixel within the cookie of the currently used main chunky Shape to /represent/ the specified colour or 0 in the Shape's graphic. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the cookie.

1.108 mpoint

```
MPoint (Xpos.w,Ypos.w[,BitmapToRead.w]) ; Return the colour of a single pixel in a ↩
     bitmap
```

Returns the colour (register number) of a single pixel in the currently used chunky Bitmap, or the specified chunky Bitmap.

1.109 mpointshape

```
MPointShape (Xpos.w,Ypos.w[,ShapeToRead.w]) ; Return the colour of a single pixel ↩
    in a shape
```

Returns the colour (register number) of a single pixel in the currently used chunky Shape, or the specified chunky Shape.

1.110 mpointstencil

```
MPointStencil (Xpos.w,Ypos.w[,BitmapToRead.w]) ; Return the status of a single ↩
    pixel in a stencil. -1=Data, 0=Background
```

Returns True or False representing wether the colour (register number) of a single pixel in the currently used or specified chunky Bitmap is transparent or not, based on the Bitmap's stencil.

1.111 mpointcookie

```
MPointCookie (Xpos.w,Ypos.w[,ShapeToRead.w]) ; Return the status of a single pixel ↩
    in a cookie. -1=Data, 0=Background
```

Returns True or False representing wether the colour (register number) of a single pixel in the currently used or specified chunky Shape is transparent or not, based on the Shape's cookie.

1.112 msscroll

```
MSScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[,CustomOffsets.l ↩
   ]] ; Copy bm 2 bm and st 2 st
```

Stencil-Scrolls an area within the currently used chunky Bitmap, or using a different source Bitmap if specified. If SScrollCut is Off, a normal 'SBlit' will be performed in which the source Bitmap's graphic will be copied to the destination Bitmap's graphic, and the source Bitmap's stencil is copied to the destination Bitmap's stencil. If SScrollCut is On, the destination Bitmap's stencil will be used to protect areas of the destination Bitmap's graphic, instead of copying the source stencil into the destination stencil, effectively scrolling the source image 'behind' the protected parts of the destination. The destination Bitmap should have a stencil. If CustomOffsets.l is specified then a list of custom values will be used for each row of the scroll. Please refer to MScroll for detailed information on the list format and what it does.

1.113 msscrollshape

```
MSScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
   CustomOffsets.l]] ; Copy sh2sh and ck2ck
```

Does the same as MSScroll except with regards to chunky Shapes. The Shape's graphic and cookie are copied to the destination Shape's graphic and cookie, or the Shape's graphic is cut 'behind' the cookie-protected areas of the destination Shape's image.

1.114 msscrollbitmaptoshape

```
MSScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w[, ↩
   CustomOffsets.l]];bm2shandst2ck
```

Does the same as MSScroll except that the source is a chunky Bitmap and the destination is a chunky Shape. The Bitmap's graphic and stencil are copied to the destination Shape's graphic and cookie, or the Bitmap's graphic is cut 'behind' the cookie-protected areas of the destination Shape's image.

1.115 msscrollshapetobitmap

```
MSScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w[, ↩
    CustomOffsets.l]]; sh2bmandck2st
```

Does the same as MSScroll except that the source is a chunky Shape and the destination is a chunky Bitmap. This is effectively the equivelent of an 'SBlit' of a Shape to a Bitmap. The Shape's graphic and cookie are copied to the destination Bitmap's graphic and stencil, or the Shape's graphic is cut 'behind' the stencil-protected areas of the destination Bitmap's image.

1.116 msmaskscroll

```
MSMaskScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Stencil- ↩
    Copy bm 2 bm and st 2 st
```

Stencil-MaskScrolls an area within the currently used chunky Bitmap, or using a different source Bitmap if specified. Width should be a multiple of 4. If SScrollCut is Off, a normal 'SMaskBlit' will be performed in which the source Bitmap's graphic will be mask-copied to the destination Bitmap's graphic (with masking), and the source Bitmap's stencil is copied to the destination Bitmap's stencil (with masking). If SScrollCut is On, the destination Bitmap's stencil will be used to protect areas of the destination Bitmap's graphic, instead of copying the source stencil into the destination stencil, effectively scrolling the source image 'behind' the protected parts of the destination and yet with masking to cause any areas in the source graphic of Colour 0 to be used as transparent. Both source and destination Bitmaps should have stencils.

1.117 msmaskscrollshape

```
MSMaskScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Cookie ↩
    -Copy sh2sh and ck2ck
```

Does the same as MSMaskScroll except with regards to chunky Shapes. The Shape's graphic and cookie are mask-copied to the destination Shape's graphic and cookie, or the Shape's graphic is mask-cut 'behind' the cookie-protected areas of the destination Shape's image.

1.118 msmaskscrollbitmaptoshape

```
MSMaskScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ↩
    ;Sten-Copy bm2sh&st2ck
```

Does the same as MSMaskScroll except that the source is a chunky Bitmap and the destination is a chunky Shape. The Bitmap's graphic and stencil are mask-copied to the destination Shape's graphic and cookie, or the Bitmap's graphic is mask-cut 'behind' the cookie-protected areas of the destination Shape's image.

1.119 msmaskscrollshapetobitmap

```
MSMaskScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ↩
    ; Cook-Copy sh2bm&ck2st
```

Does the same as MSMaskScroll except that the source is a chunky Shape and the destination is a chunky Bitmap. This is effectively the equivelent of an 'SMaskBlit' of a Shape to a Bitmap. The Shape's graphic and cookie are mask-copied to the destination Bitmap's graphic and stencil, or the Shape's graphic is mask-cut 'behind' the stencil-protected areas of the destination Bitmap's image.

1.120 msblockscroll

```
MSBlockScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Block- ↩
    Copy bm 2 bm and st 2 st
```

Stencil-BlockScrolls an area within the currently used chunky Bitmap, or using a different source Bitmap if specified. Width, X1 and X2 should be multiples of 16. If SScrollCut is Off, a normal 'SBlockBlit' will be performed in which the source Bitmap's graphic will be mask-blockcopied to the destination Bitmap's graphic (with masking), and the source Bitmap's stencil is copied to the destination Bitmap's stencil (with masking). If SSCrollCut is On, the destination Bitmap's stencil willbe used to protect areas of the destination Bitmap's graphic, instead of copying the source stencil into the destination stencil, effectively scrolling the source image 'behind' the protected parts of the destination. Both source and destination Bitmaps should have stencils. 040+ routines are used where possible but not when MSScrollCut is On as this is not possible.

1.121 msblockscrollshape

```
MSBlockScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Block ↩
    -Copy sh2sh and ck2ck
```

Does the same as MSBlockScroll except with regards to chunky Shapes. The chunky Shape's graphic and cookie are block-blitted to the destination Shape's graphic and cookie, or the source Shape's graphic is block-cut 'behind' the cookie-protected areas of the destination Shape.

1.122 msblockscrollbitmaptoshape

```
MSBlockScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w ↩
    ];BlockCopy bm2sh&st2ck
```

Does the same as MSBlockScroll except that the source is a chunky Bitmap and the destination is a chunky Shape. The chunky Bitmap's graphic and stencil are block-blitted to the destination Shape's graphic and cookie, or the source Bitmap's graphic is block-cut 'behind' the cookie-protected areas of the destination Shape.

1.123 msblockscrollshapetobitmap

```
MSBlockScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w ↩
    ];BlockCopy sh2bm&ck2st
```

Does the same as MSBlockScroll except that the source is a chunky Shape and the destination is a chunky Bitmap. The chunky Shape's graphic and cookie are block-blitted to the destination Bitmap's graphic and stencil, or the source Shape's graphic is block-cut 'behind' the stencil-protected areas of the destination Bitmap.

1.124 msscrollcut

```
MSScrollCut On/Off ; 0=Paste stencil/cookie, <>0=Cut using stencil/cookie
```

Sets the 'cut' mode of the MSScroll family of scroll's On or Off. If On, any calls the Stencil-Scrolls will cause the source image to be placed behind the stencil or cookie protected areas of the destination. If SScrollCut is switched off it will cause the Stencil-Scrolls to perform the same operation on the stencils and cookies as is performed on the images and no cutting/protecting will be performed, which corresponds to a normal 'SBlit'.

1.125 museshapebank

```
MUseShapeBank BankNumber.w ; Current shape bank, 0..31
```

In Mildred there are currently 32 built-in 'banks' of Shapes. When you reserve some shapes with MReserveShapes they usually only use bank 0, but if you wish you can use *seperate* sets of shapes in banks numbered 0 to 31. You should 'use' a bank before reserving shapes. Note that operations accross banks are not possible and a current Shape number of 5 means chunky Shape 5 in the 'current' bank.

1.126 mpicturedissolvein

```
MPictureDissolveIn PictureBitmapNum.w,Colour.b ; Do a picture-based colour-number ↩
    dissolve-in of a bitmap
```

Performs a picture-based dissolve in order to 'bring in' an source image onto a destination bitmap. You first have to 'use' two bitmaps with MUseBitmap, in which you specify the source and destination chunky Bitmap's. Then you specify the 'Picture Bitmap' as a parameter to this command which will be the number of a chunky Bitmap containing an 'effect' picture. The routine will search for the location of pixels in the effect picture that are of the specified colour. When it finds one it will copy a pixel from those coordinates in the source Bitmap to the same coordinates in the destination Bitmap. Note that all three Bitmaps in this operation have to be the same size. To fully bring-in a whole picture you should call this command as many times as there are colours in the effect-picture so that all pixels are eventually copied. The more colours you make use of in the effect-picture the longer the dissolve will take. Note that the effect-picture

should still be 256-colours even if you only use 64, for example. To create an effect-picture that produces a nice screenwipe you should create a simple greyscale gradient in something like Deluxe Paint and then perform some good image-processing operations on it in something like ImageFX to distort and mangle the gradient to bring variety into the effect. Note that if the destination Bitmap already contains an image and you are dissolving in a new one, the two images should use the same colour palette otherwise one of the two images will appear in the wrong colours. If the destination bitmap starts off blank it will just cause the source image to gradually appear on, for example, a black background.

1.127 mmaskscrollmode

```
MMaskScrollMode [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
    MColourMode/MReMapMode/MSimpleReMapMode
```

The Mask family of Scroll commands can have a 'type' of blit operation which is set with this command. You can use the already existing Blitz functions or values 0, 1, 2 and 3 which represent CookieMode, EraseMode, InvMode and SolidMode. The normal mode is CookieMode in which the source's stencil or cookie is used to perform a blit in which areas of Colour 0 are transparent. EraseMode causes the source's stencil or cookie to erase an area matching its shape on the destination image. InvMode causes the destination to be inverted in the shape of the source stencil or cookie, and SolidMode places a solid copy of the source's stencil or cookie into the destination's image, which will appear as Colour 255. If used as a function, MMaskScrollMode will return a number representing the current mode.

1.128 mblitmode

```
MBlitMode [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/MColourMode/ ↩
    MReMapMode/MSimpleReMapMode
```

The normal 'Blit' family of commands can have a 'type' of blit operation which is set with this command. You can use the already existing Blitz functions or values 0, 1, 2 and 3 which represent CookieMode, EraseMode, InvMode and SolidMode. The normal mode is CookieMode in which the source's cookie is used to perform a blit in which areas of Colour 0 are transparent. EraseMode causes the source's cookie to erase an area matching its shae on the destination image. InvMode causes the destination to be inverted in the shape of the source cookie, and SolidMode places a solid copy of the cookie into the destination's image, which will appear as colour 255. MBlitMode also works with MColourMode, MReMapMode and MSimpleReMapMode. If MBlitMode is used as a function it will return a number representing the current mode.

1.129 mblit

```
MBlit [ShapeNumber.w,]Xpos.w,Ypos,w ; Blit shape to bitmap, any coords
```

Performs a straight masked-blit of a chunky Shape into a chunky Bitmap to

coordinates Xpos,Ypos. If the ShapeNumber is omitted the currently used Shape is used. The type of blit performed is defined with the MBlitMode command. Normally the Shape is simply cookie-blitted onto the Bitmap, with areas of Colour 0 as transparent. If the Shape does not have a cookie, it will be blitted in unmasked 'Replace' mode.

1.130 mblock

```
MBlock [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit shape to bitmap, align Xpos and ↩
    width in multiples of 16!
```

Performs a straight unmasked block-blit of a chunky Shape into a chunky Bitmap to coordinates Xpos,Ypos. Xpos should be a multiple of 16 and so should the width of the Shape. If the ShapeNumber is omitted the currently used Shape is used. This command performs only a solid 'replace' blit and does not use Colour 0 as transparent.

1.131 mtile16x16

```
MTile16x16 [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape to bitmap, size ↩
    must be 16x16, align x/y
```

Performs a highly optimised blit of the currently-used or specified chunky Shape into the currently used chunky Bitmap at Xpos,Ypos. The Shape should be 16x16 in size and *both* Xpos and Ypos should be multiples of 16.

1.132 mtile32x32

```
MTile32x32 [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape to bitmap, size ↩
    must be 32x32, align x/y
```

Performs a highly optimised blit of the currently-used or specified chunky Shape into the currently used chunky Bitmap at Xpos,Ypos. The Shape should be 32x32 in size and *both* Xpos and Ypos should be multiples of 32.

1.133 mstile16x16

```
MSTile16x16 [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape&cook 2 bitmap, ↩
    size 16x16, align x/y
```

Performs a highly optimised stencil-blit of the currently-used or specified chunky Shape to the currently used chunky Bitmap at Xpos,Ypos. The Shape should be 16x16 in size and *both* Xpos and Ypos should be multiples of 16. The Shape's graphic will be blitted to the Bitmap's graphic, and the Shape's cookie will be blitted to the Bitmap's stencil.

1.134 mstile32x32

```
MSTile32x32 [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape&cook 2 bitmap, ↩
    size 32x32, align x/y
```

Performs a highly optimised stencil-blit of the currently-used or specified chunky Shape to the currently used chunky Bitmap at Xpos,Ypos. The Shape should be 32x32 in size and *both* Xpos and Ypos should be multiples of 32. The Shape's graphic will be blitted to the Bitmap's graphic, and the Shape's cookie will be blitted to the Bitmap's stencil.

1.135 mstile16x16store

```
MSTile16x16Store [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape&cook 2 ↩
    bitmaps, size 16x16, align x/y
```

Does the same as MSTile16x16 except that the Shape's graphic will *also* be placed onto the *second* currently-used chunky Bitmap's image. This is mainly so that a background-store Bitmap can be created for use with Bitmap-based un-queue operations.

1.136 mstile32x32store

```
MSTile32x32Store [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape&cook 2 ↩
    bitmaps, size 32x32, align x/y
```

Does the same as MSTile32x32 except that the Shape's graphic will *also* be placed onto the *second* currently-used chunky Bitmap's image. This is mainly so that a background-store Bitmap can be created for use with Bitmap-based un-queue operations.

1.137 mtile16x16store

```
MTile16x16Store [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 16x16 shape to 2 ↩
    bitmaps, size 16x16, align x/y
```

Does the same as MTile16x16 except that the Shape's graphic will *also* be placed onto the *second* curretly-used chunky Bitmap's image. This is mainly so that a background-store Bitmap can be created for use with Bitmap-based un-queue operations. The Shape's cookie is not copied anywhere.

1.138 mtile32x32store

```
MTile32x32Store [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit 32x32 shape to 2 ↩
    bitmaps, size 32x32, align x/y
```

Does the same as MTile32x32 except that the Shape's graphic will *also* be placed

onto the *second* currently-used chunky Bitmap's image. This is mainly so that a background-store Bitmap can be created for use with Bitmap-based un-queue operations. The Shape's cookie is not copied anywhere.

1.139 mreservequeues

MReserveQueues [(]NumberOfQueues.w[)] ; Reserve structure-memory for Queues

Before using any of the queue-related commands you need to call MReserveQueues in order to allocate space to store the basic information needed for some chunky Queue objects, unless making use of the default reserved Queue objects. This does not allocate any space for storing actual items onto the Queue, for which you need to first create a chunky Queue object. Once reserved, Queue numbers will range from 0 upwards. If used as a function, this command will return the address in memory where the chunky Queue structures are stored, or 0 for failure.

1.140 mfreequeue

MFreeQueue [FirstQueue.w[,LastQueue.w]] ; Free a Queue, a range of queues, or all queues

Frees a specific, all or a range of previously-created queues. If the parameters are omitted it will free all created chunky Queue objects.
If you specifiy a second parameter a range of queues willbe freed. When freed, a Queue will be 'killed' and the memory for the Queue items freed so the Queue is effectively empty. If the specified Queues don't exist it will report an error, but will not report an error when freeing all Queues.

1.141 maddrqueue

MAddrQueue [(QueueNumber.w)] ; Returns address of Queue structure

Returns the address of a specified (or currently used) chunky Queue object. This is the address of the Queue structure, not the address of the item data. If a Queue number is not specified then the currently used queue will be referenced.

1.142 mqueue

MQueue [(]QueueNumber.w,NumberOfItems.w[)] ; Allocmem for Queue list items

Creates a new chunky Queue object. The Queue is initialised so will be empty. Once created, you can start to add items to the Queue using the various 'Q' blits. Only as many items as you specify are allowed to be added to the Queue before it will become full. If used as a function, this command will return the address at which the chunky Queue structure is stored, or 0 for failure.

1.143 mflushqueue

```
MFlushQueue QueueNumber.w ; Empties the queue to contain no items
```

Flushes the entire contents of the specified chunky Queue object, making it empty. Any following un-queue operations will have no effect and you can start adding new items to the Queue again.

1.144 mqblitmode

```
MQBlitMode [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/MColourMode/ ↩
    MReMapMode/MSimpleReMapMode
```

The normal 'QBlit' family of commands can have a 'type' of blit operation which is set with this command. You can use the already existing Blitz functions or values 0, 1, 2 and 3 which represent CookieMode, EraseMode, InvMode and SolidMode. The normal mode is CookieMode in which the source's cookie is used to perform a blit in which areas of Colour 0 are transparent. EraseMode causes the source's cookie to erase an area matching its shae on the destination image. InvMode causes the destination to be inverted in the shape of the source cookie, and SolidMode places a solid copy of the cookie into the destination's image, which will appear as colour 255. The QBlitMode only affects the blit operation, not the un-queue. MQBlitMode also works with MColourMode, MReMapMode and MSimpleReMapMode. If MQBlitMode is used as a function it will return a number representing the current mode.

1.145 mautousequeues

```
MAutoUseQueues True/False ; Automatically 'use' new Queues. <>0=True
```

If set to True/On, any chunky Queue objects created in future will automatically be 'used' as the 'current' main chunky Queue object. Other commands can then use this to 'assume' which Queue you are referring to so that you don't have to keep specifying it each time.

1.146 musequeue

```
MUseQueue MainQueueNum.w[,SecondQueueNum.w[,ThirdQueueNum.w]] ; Current to use
```

Selects which chunky Queue object to use as 'current', and also second and third 'current' Queues. These current Queues will be assumed later so that you don't have to keep specifying which chunky Queue object(s) to use. Usually only the main (first) Queue is used.

1.147 musedqueue

```
MUsedQueue  ; Returns currently used Queue
```

Returns the number of the currently used main chunky Queue object.

1.148 mqblit

```
MQBlit [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos,w ; QBlit shape to bitmap, any coords
```

Performs a straight masked-blit of a chunky Shape into a chunky Bitmap to coordinates Xpos,Ypos. If the ShapeNumber is omitted the currently used Shape is used. The type of blit performed is defined with the MQBlitMode command. Normally the Shape is simply cookie-blitted onto the Bitmap, with areas of Colour 0 as transparent. Just prior to the blit being performed an entry is added to the specified (or assumed) chunky Queue object corresponding to the area that the Shape has changed in the destination Bitmap. This area can later be cleared or restored using an un-queue.

1.149 mqblock

```
MQBlock [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos.w ; QBlock-blit shape 2 bitmap,  ↩
    align Xpos & width in mult of 16
```

Performs a straight unmasked block-blit of a chunky Shape into a chunky Bitmap to coordinates Xpos,Ypos. Xpos should be a multiple of 16 and so should the width of the Shape. If the ShapeNumber is omitted the currently used Shape is used. This command performs only a solid 'replace' blit and does not use Colour 0 as transparent. Just prior to the blit being performed an entry is added to the specified (or assumed) chunky Queue object corresponding to the area that the Shape has changed in the destination Bitmap. This area can later be cleared or restored using an un-queue.

1.150 munqueue

```
MUnQueue QueueNumber.w[,FirstItem.w,LastItem.w][,BitmapNumber.w] ; UnQueue [range  ↩
    of] queued objects [&flush]
```

```
Un-Queues the specified Queue to the current chunky Bitmap. If the Bitmap  ↩
    parameter
```

is omitted a 'clearscreen' operation will be performed (to current ink) in all of the areas that have been recorded into the Queue list. If the Bitmap parameter is specified a chunky Bitmap will be used as a background store, from which the areas recorded in the Queue will be blitted to the currently used Bitmap instead of just performing a clearscreen. Once all items have been un-queued, the Queue will be completely flushed. If FirstItem and LastItem are specified, a range of items in the queue will be unqueued. However, in this case the queue will NOT be flushed, and an MFlushQueue will likely be needed at a later stage.

1.151 mbitmapptr

```
MBitmapPtr [Xpos.w,Ypos.w][,BitmapNumber.w] ; Return data address calculated using ↩
    bitmap [and coords]
```

Returns an address based on the specified chunky Bitmap object. If a Bitmap object is not specified, the currently-used object will be used. The address is calculated by taking the base address of the Bitmap's graphics data. If the wrapping feature is switched on for the Bitmap, the Bitmap's origin/handle will cause an offset to be added to the address to represent those coordinates. If the clipping feature is switched on for the Bitmap, the top-lefthand coordinates of the Bitmap's clip window will be added to the address also to provide an offset representing that location. Finally, if you have specified Xpos and Ypos these coordinates will also be added to the address to represent the memory location of that location. The final address is then returned and is of particular use for the 'Chunky.l' parameter of Mc2p.

1.152 mshapeptr

```
MShapePtr [Xpos.w,Ypos.w][,ShapeNumber.w] ; Return data address calculated using  ↩
   shape [and coords]
```

Does the same as MBitmapPtr except with regards to a chunky Shape object or the currently used chunky Shape object. The address returned is based on the Shape's graphics data and possibly handle, clip window and specified Xpos,Ypos coords. This command can be used to allow you to easily view a Shape's graphic with the c2p system.

1.153 mstencilptr

```
MStencilPtr [Xpos.w,Ypos.w][,BitmapNumber.w] ; Return address calculated using  ↩
   stencil [and coords]
```

Does the same as MBitmapPtr except with regards to the stencil of a chunky Bitmap object or the stencil of the currently used chunky Bitmap object. The address returned is based on the Bitmap's stencil data and possibly handle/origin, clip window and specified Xpos,Ypos coords. This command can be used to allow you to easily view a Bitmap's stencil with the c2p system.

1.154 mcookieptr

```
MCookiePtr [Xpos.w,Ypos.w][,ShapeNumber.w] ; Return address calculated using  ↩
   cookie [and coords]
```

Does the same as MBitmapPtr except with regards to the cookie of a chunky Shape object or the cookie of the currently used chunky Shape object. The address returned is based on the Shape's cookie data and possibly handle/origin, clip window and specified Xpos,Ypos coords. This command can be used to allow you to easily view a Shape's cookie with the c2p system.

1.155 mqdummy

```
MQDummy [Queue.w,]Xpos.w,Ypos.w,Width.w,Height.w ; Add an item to a queue without ↩
    having to do a blit
```

Adds an item to the specified or currently used chunky Queue object, without having to perform a blit operation. Width should be a multiple of 4. This command allows you to add 'items' to the queue to represent other kinds of blits which are not directly supporting of the Queue system, such as the Tile and Scroll routines.

1.156 msblitmode

```
MSBlitMode [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/MColourMode/ ↩
    MReMapMode/MSimpleReMapMode
```

The normal 'SBlit' family of commands can have a 'type' of blit operation which is set with this command. You can use the already existing Blitz functions or values 0, 1, 2 and 3 which represent CookieMode, EraseMode, InvMode and SolidMode. The normal mode is CookieMode in which the source's cookie is used to perform a blit in which areas of Colour 0 are transparent. EraseMode causes the source's cookie to erase an area matching its shae on the destination image. InvMode causes the destination to be inverted in the shape of the source cookie, and SolidMode places a solid copy of the cookie into the destination's image, which will appear as colour 255. The type of blit will be performed on both the destination graphic and the destination stencil if applicable. MSBlitMode also works with MColourMode, MReMapMode and MSimpleReMapMode. If MSBlitMode is used as a function it will return a number representing the current mode.

1.157 msblit

```
MSBlit [ShapeNumber.w,]Xpos.w,Ypos,w ; Blit shape to bitmap and cookie to stencil, ↩
     any coords
```

Performs a straight masked-blit of a chunky Shape into a chunky Bitmap to coordinates Xpos,Ypos. If the ShapeNumber is omitted the currently used Shape is used. The type of blit performed is defined with the MSBlitMode command. Normally the Shape is simply cookie-blitted onto the Bitmap, with areas of Colour 0 as transparent. If SBlitCut is Off, the same operation is performed with the Shape's cookie, in which the cookie is blitted to the Bitmap's stencil using the same SBlitMode. If SBlitCut is On, the Bitmap's stencil will act to protect areas of the Bitmap's image, causing the Shape to be blitted 'behind' the protected areas. With SBlitCut On the cookie is no longer copied to the stencil.

1.158 msblock

```
MSBlock [ShapeNumber.w,]Xpos.w,Ypos.w ; Block-blit shape to bitmap & cookie 2  ↩
    stencil, Xpos&Width in 16's
```

Performs a straight unmasked-blit of a chunky Shape into the chunky Bitmap to coordinates Xpos,Ypos. Xpos should be a multiple of 16 and the shape should also have a Width that is multiple of 16. The blit will be an un-masked blit and will not be affected by MSBlitMode but it is possible to cut-blit the block-type Shape 'behind' the Bitmap's stencil using MSBlitCut On. Normally the Shape's cookie will also be block-blitted to the Bitmap's stencil.

1.159 msblitcut

MSBlitCut On/Off ; 0=Paste stencil/cookie, <>0=Cut using stencil/cookie

Selects wether to perform a stencil-cut or a normal stencil-blit when performing MSblit and MSBlock. If SBlitCut is Off, a Shape's graphic is blitted to the Bitmap's graphic and the Shape's cookie is blitted to the Bitmap's stencil. If SBlitCut is On, the destination Bitmap's stencil will be used to protect areas of the Bitmap's graphic rather than copying the cookie to the stencil, which has the effect of looking as thought the operation is performed 'behind' the protected areas.

1.160 mqsblitmode

MQSBlitMode [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/MColourMode/ ↩
MReMapMode/MSimpleReMapMode

Does the same as MSBlitMode except with reference to the QS-type blits. If used as a function it will return a number representing the current mode.

1.161 mqsblit

MQSBlit [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos,w ; QBlit shape to bitmap and cookie ↩
to stencil, any coords

Does the same as MSBlit except that an item is also added to the specified or currently used chunky Queue object to represent the area that the blit has changed on the Bitmap's graphic. Note that areas of the Bitmap's stencil are not accessible by the queue system so when performing an SBlit or SBlock with SBlitCut Off, changes made to the Bitmap's stencil will be permanent.

1.162 mqsblock

MQSBlock [[Queue.w,]ShapeNumber.w,]Xpos.w,Ypos.w ; QBlock-blit shape 2 bitmap, ↩
Xpos&width mult of 16

Does the same as MSBlock except that an item is also added to the specified or currently used chunky Queue object to represent the area that the blit has changed on the Bitmap's graphic. Note that areas of the Bitmap's stencil are not accessible by the queue system so when performing an SBlit or SBlock with SBlitCut Off, changes made to the Bitmap's stencil will be permanent.

1.163 mqsblitcut

```
MQSBlitCut On/Off ; 0=Paste stencil/cookie, <>0=Cut using stencil/cookie. Adds ↩
    entry to queue
```

Does the same as MSBlitCut but with reference to the QS-type blits.

1.164 mboxf

```
MBoxF Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a bitmap [to ↩
    specified colour]
```

Fills the specified area within the currently used chunky Bitmap to the currently used ink colour or the specified colour. Width of the box does NOT have to be a multiple of 4 and the box should be at least 1x1 in size. The specified area should fit within the dimensions of the Bitmap, and it is possible for X2,Y2 to be further left and/or further up the screen than X1,Y1. See also MDrawingMode for further details of the drawing modes that are available.

1.165 mboxfshape

```
MBoxFShape Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a shape [to ↩
     specified colour]
```

Does the same as MBoxF but draws the filled box to the currently used chunky Shape.

1.166 mboxfstencil

```
MBoxFStencil Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a stencil ↩
     [to specified colour]
```

Does the same as MBoxF but draws the filled box to the stencil of the currently used chunky Bitmap. The actual value drawn to the stencil /represents/ the currently used ink colour or the specified colour. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the stencil.

1.167 mboxfcookie

```
MBoxFCookie Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw a filled box in a cookie [ ↩
    to specified colour]
```

Does the same as MBoxF but draws the filled box to the cookie of the currently used chunky Shape. The actual value drawn to the cookie /represents/ the currently used ink colour or the specified colour. If the drawing mode is

set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the cookie.

1.168 mbox

MBox Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a bitmap [to specified colour]

Draws an unfilled box according to the specified area within the currently used chunky Bitmap to the currently used ink colour or the specified colour. Width of the box does NOT have to be a multiple of 4 and the box should be at least 1x1 in size. The specified area should fit within the dimensions of the bitmap. X2,Y2 is allowed to be further left or further up the screen than X1,Y1. See also MDrawingMode for a description of the drawing modes available.

1.169 mboxshape

MBoxShape Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a shape [to specified colour]

Does the same as MBox but draws the unfilled box to the currently used chunky Shape.

1.170 mboxstencil

MBoxStencil Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a stencil [to specified colour]

Does the same as MBox but draws the unfilled box to the stencil of the currently used chunky Bitmap. The actual value drawn to the stencil /represents/ the currently used ink colour or the specified colour. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the stencil .

1.171 mboxcookie

MBoxCookie Xpos.w,Ypos.w,Xpos2.w,Ypos2.w[,Colour] Draw an unfilled box in a cookie [to specified colour]

Does the same as MBox but draws the unfilled box to the cookie of the currently used chunky Shape. The actual value drawn to the cookie /represents/ the currently used ink colour or the specified colour. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the cookie.

1.172 mplanar16tobitmap

```
MPlanar16ToBitmap BitmapNum.w,PlanarAddr.l[,OpWidth.w,OpHeight.w,PlanarWidth.w, ↩
    PlanarHeight.w] ; Convert p2c
```

Converts an area of planar-format graphics data into a previously-created chunky Bitmap object's graphic. The planar graphic should have contigous non-interleaved bitplanes without a linemodulo and should have a width that is a multiple of 16. If the destination chunky Bitmap is larger than the operation, or the source planar 'Bitmap' is larger than the operation, then you should use the long version of this command and again the same rules about contiguous bitmaps and so on apply. The operation width should be a multiple of 16, due mainly to the way that the conversion handles words of planar data at a time (a word is 16 pixels in planar). You should use PlanarAddr.l to specify the location of the topleft corner of the planar graphic. If all goes well, the planar graphic (perhaps held in a blitz bitmap object cludged onto pre-reserved contigous memory) will be converted to chunky format and placed into the graphic of the specified chunky Bitmap object.

1.173 mplanar16toshape

```
MPlanar16ToShape ShapeNum.w,PlanarAddr.l[,OpWidth.w,OpHeight.w,PlanarWidth.w, ↩
    PlanarHeight.w] ; Convert p2c
```

Does the same as MPlanar16ToBitmap except that the converted data is output to a previously-created chunky Shape object rather than a chunky Bitmap.

1.174 mgenericptr

```
MGenericPtr Xpos.w,Ypos.w,BaseAddress.l,RowWidth.w ; Calculate and return address  ↩
    based on inputs
```

Returns an address based on the parameters specified. Xpos,Ypos are coordinates which may represent the offset within a bitmap or something. The expression for working out the value to return is Base=BaseAddress+(Ypos*RowWidth)+Xpos. RowWidth is therefore needed to work out the width of the image for creating the proper offset for the Xpos,Ypos coordinate. If using this command with regards to planar bitmaps (blitz's normal Bitmap objects) for example, you would need to devide Xpos and RowWidth by 8 before passing the parameters.

1.175 mcludgecookie

```
MCludgeCookie ShapeNumber.w,Memory.l ; Cludge shape's cookie from existing mem
```

Once a chunky Shape object has been created you may wish to cludge a new cookie from some existing graphics memory rather than create a new memory area for it. This command allows you to do that. Your cookie data should be aligned to a multiple of 16 bytes in memory. This is to provide proper support for block and tile routies on 040+ cpu's. Please note to take care in cludging cookie data that

it shares the same linemodulo as the main image data. That is, if you cludged the shape onto existing memory, or created it as a window within another shape/bitmap and the width of the new shape is smaller than the original data, there will be a 'leftover' amount of bytes that have to be skipped each line. This amount is the linemodulo, and in order to operate correct Mildred requires that your cludged cookie data is created under similar circumstances, ie sharing the same linemodulo. It is therefore in appropriate, for example, to make a shape using MBitmapShape for a small area within a large bitmap, the cookie for that shape must also be cludged within memory that is structured similarly to the main data - ie taking into account any linemodulo the shape data has. Note, therefore, that cludging cookies for shapes with linemodulo's larger than 0 must be handled carefully, otherwise any masked operations/blits using the shape will not behave as they should.

1.176 mcludgestencil

```
MCludgeStencil BitmapNumber.w,Memory.l ; Cludge bitmap's stencil from existing mem
```

Does the same as MCludgeCookie except with regards to cludging a new stencil for a previously-created chunky Bitmap.

1.177 mremap

```
MReMap [Colour#0.b,Colour#1.b,BitmapNum.w] *or* [RemapTable.l[,BitmapNum.w]] ;  ↩
    Remap #0 to #1 or with table
```

Remaps the colours in a chunky Bitmap's graphic from one colour to another. You specify two colour numbers. Pixels of the first colour number (0..255) are searched for and when found a replaced with the second colour number (0..255). So all pixels of the first specified colour become the second specified colour. There is an alternative set of parameters possible with this command in which you specify the address of a remapping table and possibly also the number of a Bitmap to use (if not specified it will use the currently used chunky Bitmap). The remapping table should be 256 bytes, each byte corresponding to the 256 colours of the graphic in the chunky Bitmap. The first colour value is read from the Bitmap using a kind of 'point' operation. The second colour value, which is the colour it will be changed to, is then read from the remapping table relevant to the colour that had been found in the Bitmap. ALL pixels in the Bitmap will be remapped, although this can be cropped using the Bitmap's clip window with clipping set to On for that Bitmap. You currently need to generate the remapping table yourself.

1.178 mremapshape

```
MReMapShape [Colour#0.b,Colour#1.b,ShapeNum.w] *or* [RemapTable.l[,ShapeNum.w]] ;  ↩
    Remap #0 to #1 or with table
```

Does the same as MReMap but performs the remapping operation on the currently used or specified chunky Shape object.

1.179 mline

```
MLine [Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b] ;Draw a line from X1,Y1 to X2,Y2 ↩
    in a Bitmap, Colour or 0
```

Draws a line in the specicied colour or ink, starting at Xpos,Ypos and ending at Xpos2,Ypos2 in the currently used chunky Bitmap. If Xpos,Ypos are omitted, the end coordinates of the previously drawn line will be used, allowing you to easily draw a chain of connected lines (a polyline). See also MDrawingMode for further details of the drawing modes that are available.

1.180 mlineshape

```
MLineShape [Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b] ;Draw a line from X1,Y1 to ↩
    X2,Y2 in a Shape, Colour or 0
```

Does the same as MLine, except that the line is rendered to the currently used chunky Shape.

1.181 mlinestencil

```
MLineStencil [Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b];Draw a line from X1,Y1 to ↩
    X2,Y2 in a stencil, Col or 0
```

Does the same as MLine, except that the line is drawn to the stencil of the currently used chunky Bitmap, and the colour drawn will /represent/ the specified colour or ink. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the stencil.

1.182 mlinecookie

```
MLineCookie [Xpos.w,Ypos.w,]Xpos2.w,Ypos2.w[,Colour.b] ;Draw a line from X1,Y1 to ↩
    X2,Y2 in a cookie, Col or 0
```

Does the same as MLine, except that the line is drawn to the cookie of the currently used chunky Shape, and the colour drawn will /represent/ the specified colour or ink. If the drawing mode is set to MReMapMode or MSimpleReMapMode with the MDrawingMode command, MColourMode will be used instead as it is illogical to try to perform remapping to the cookie.

1.183 mink

```
MInk MainColour.b[,SecondColour.b[,ThirdColour.b] ; Set what colour to assume as ↩
    currently used. 0..255
```

Sets the currently used colour so that graphics operations can assume it rather than you having specify a colour for every call. The colour should be from 0 to 255 inclusive. The ink colour will be used for a variety of operations, but note that clearscreen operations (MCls) will always assume a colour of 0 as that is more logical. The ink colour, prior to being defined by the programmer, will default to 1. If you specify SecondColour and possible ThirdColour these will be stored as the secondary and third colours to assume, although will not be used in most operations.

1.184 mcolourmode

```
MColourMode ;Returns value 4 which represents 'colour' mode in the blit modes
```

For use with one of the 'BlitMode' commands, such as MBlitMode, the MColourMode function can be used in the same way as the default CookieMode, SolidMode, InvMode and EraseMode. MColourMode is a new blit mode that will render the source image to the destination so that all of the pixels in the source are of one colour. That colour is specified with the MInk command. The colour-blit is based on the shape's cookie which is turned into pixels of the ink colour and then output. MColourMode performs a similar function to SolidMode, except that you can define what colour the shape will be solidly displayed in. MColourMode can be used with MBlitMode, MQBlitMode, MSBlitMode, MQSBlitMode and MMaskScrollMode.

1.185 mreservetables

```
MReserveTables [()NumberOfTables.w[)] ; Reserve structure-memory for Tables
```

Allows you to reserve space for the structures of some table objects. It does not allocate space for the actual table itself. A table object is similar to a queue but generally contains a lookup table of byte data rather than information about the position of areas to be unqueued. By default, 20 tables will be allocated, meaning that at most you are allowed to create tables numbering 0 to 19. If you want more tables than this then you need to use MReserveTables. Any pre-existing tables will be erased before space for new tables is made. If used as a function, this command will return the address in memory at which the table structures have been allocated.

1.186 mfreetable

```
MFreeTable [FirstTable.w[,LastTable.w]] TableNumber.w ; Free a Table, a range of ↩
    tables or all tables
```

Frees a specific, all or range of Table objects. Generally there will not be a check to see if the tables exist or not. Once freed, the Table objects will be effectively dead and free of actual data. If you specify the second parameter then a range of tables will be freed, and if you omit all parameters all of the tables will be freed.

1.187 maddrtable

```
MAddrTable [(TableNumber.w)] ; Returns address of Table structure
```

Returns the address in memory at which a particular table object is stored. This is the address of the object's structure, not the address of the data. If a Table number is not specified then the currently used table object will be referenced.

1.188 mtable

```
MTable [()TableNumber.w,SizeInBytes.l[)] ; Allocmem for Table list items
```

Initialises a new chunky Table object. The number of the table is specified with TableNumber and you should specify how many byte of memory can be stored in the table. This command is basically an elaborate form of memory allocation, but holds some data which is needed by other parts of Mildred such as the MReMap blit mode. One initialised the data in the Table will be nonsense, or possibly 0's. If used as a function, this command will return the address in memory at which the Table data is being held, which is the base address of the data.

1.189 mautousetables

```
MAutoUseTables True/False ; Automatically 'use' new Tables. <>0=True
```

Selects wether or not to automatically 'use' new Table objects as the current object. If this is set to True or On (or -1), any new Table object that is created will be automatically used as the current object. This command defaults to the 'On' state prior to alterations by the programmer.

1.190 musetable

```
MUseTable MainTableNum.w[,SecondTableNum.w[,ThirdTableNum.w]] ; Current to use
```

Selects one, two or three Table objects to use as current. Normally the MainTableNum will be used first. Once selected, other routines can assume to use these Table(s) in their operations.

1.191 musedtable

```
MUsedTable  ; Returns currently used Table
```

Returns the number of the currently used main Table object.

1.192 mtableptr

```
MTablePtr [TableNum.w] ; Returns pointer to base of the table itself
```

Returns an address in memory at which the actual data for a table is being stored. This is the base-address of the data itself, not the Table structure. If omitted, the currently used main Table will be referenced.

1.193 mremapmode

```
MReMapMode ;Returns value 5 which represents 'ReMap' mode in the blit modes (uses ↩
    current 2-dimensional table)
```

For use with one of the 'BlitMode' commands, such as MBlitMode, the MReMapMode function can be used in the same way as the default CookieMode, SolidMode, InvMode and EraseMode. MReMapMode is a new blit mode that will render the source image to the destination at the same time as performing a remapping of the colours using a previously created lookup table, ie a Table object. The currently used table object will be used and it should be a 2 dimensional table of 8-bit data, ie 256x256 bytes. When MReMapMode is the blit mode of choice, the source will be blitted to the destination in the manner according to the type of blit (MBlit, SBlit, etc), but also the source and destination will be combined using the Table to produce some kind of remapping effect. It depends on what the Table has been computed for as to what the effect will be, but it is possible to do things such as semi-transparent merges, brightness changes, add and subtract, etc. Also you should note that in MReMap mode, the entirity of the Shape's rectangle will be remapped. It is therefore necessary to build in masking support to the actual Table, by checking for combinations where the source colour is colour 0, and using a straight copy of the destination colour for those combinations. This will cause pixels of 0 in the Shape, for example, to appear to be transparent, or alternatively you can deliberately cause some effect to occur in the normally transparent areas. MReMapMode can be used with MBlitMode, MQBlitMode, MSBlitMode, MQSBlitMode and MMaskScrollMode. MReMapMode is somewhat slower than other blit modes due to the intensive processing required.

1.194 msimpleremapmode

```
MSimpleReMapMode ;Returns value 6 which is 'SimpleReMap' mode in blit modes (uses ↩
    current 1-dimensional table)
```

For use with one of the 'BlitMode' commands, as described more fully in the documentation for MReMapMode. MSimpleReMapMode is similar except that it uses a 1-dimensional remapping table, comprising just 256 bytes, each byte representing the value that the pixels in the destination (and part of the Shape's cookie) will be changed to. This simple remapping is faster than the 2-dimensional remap and does not take into account the source Shape data, except for its cookie which it uses to mask the area that will be altered in the destination. The end result is that the area in the destination, in the design of the source Shape's image, will be remapped to new values. It is quite feasible to remap only certain colours by ensuring that the colours to be protected in the table get remapped without any alteration in value. MSimpleReMapMode will work with MBlitMode, MQBlitMode, MSBlitMode, MQSBlitMode, and MMaskScrollMode.

1.195 msmaskscrollmode

```
MSMaskScrollMode [([]Mode.w[])] ; CookieMode/EraseMode/InvMode/SolidMode/ ↩
    MColourMode/MReMapMode/MSimpleReMapMode
```

The SMask family of Scroll commands can have a 'type' of blit operation which is set with this command. You can use the already existing Blitz functions or values 0, 1, 2 and 3 which represent CookieMode, EraseMode, InvMode and SolidMode. The normal mode is CookieMode in which the source's stencil or cookie is used to perform a blit in which areas of Colour 0 are transparent. EraseMode causes the source's stencil or cookie to erase an area matching its shape on the destination image. InvMode causes the destination to be inverted in the shape of the source stencil or cookie, and SolidMode places a solid copy of the source's stencil or cookie into the destination's image, which will appear as Colour 255. MSMaskScrollMode also works with MColourMode, MReMapMode and MSimpleReMapMode. If used as a function, MSMaskScrollMode will return a number representing the current mode.

1.196 mplotparticles

```
MPlotParticles CoordinateList.l,NumPoints.l[,Colour.b] ; Plot lots of points from  ↩
    a table of positions
```

Performs multiple Plot operations to the currently used chunky Bitmap, in the current Ink colour or the specified colour. CoordinateList.l should point to memory containing a list of coordinates. The coordinates should normally be within the bitmap's dimensions. The list format depents on the setting made with MParticleFormat and can be X.w,Y.w pairs, X.q,Y.q pairs, or Ptr.l actual memory addresses. The total number of points that are to be processed is passed as NumPoints.l, and notice that this is a longword value and allows for more particles than you probably have the memory capacity for, although there is a limit of about 2^29 particles. Coordinates are taken from the list and the pixels are plotted to those locations on the bitmap, in the colour being used or specified. When MParticleMode is set to MSimpleReMapMode or MReMapMode, points will be simple or complex remapped respectively. In MReMapMode, the specified colour or ink will be used as the source. In MSimpleReMapMode the destination pixels will be simple-remapped regardless of the specified or currently used colour. Please refer to MParticleFormat which you should call before performing your particle operations if you want to be sure of the list format.

1.197 mgrabparticles

```
MGrabParticles CoordinateList.l,NumPoints.l,Buffer.l ; Grab lots of points from a  ↩
    table into buffer mem
```

Performs multiple Point operations from the currently used chunky Bitmap. CoordinateList.l should point to the list of coordinates which should be in X.w,Y.w pairs, X.q,Y.q pairs or Ptr.l actual memory addresses as chosen with MParticleFormat. Specify the total number of points in NumPoints.l and finally ↩
    pass
the address of a memory buffer in Buffer.l. Pixels, which will be a byte in size

each, will be grabbed from the bitmap according to the coordinates from the list, and placed into the memory buffer. This command is mainly used for picking up areas from an image, which may be for use as part of a background-preservation system when moving using particles as an animation feature. Please refer to MParticleFormat which you should call before performing your particle operations if you want to be sure of the list format.

1.198 mdrawparticles

```
MDrawParticles CoordinateList.l,NumPoints.l,Buffer.l ; Draw lots of previously ↩
    grabbed points using a table
```

Performs a kind of 'unbuffer' using a memory buffer (Buffer.l) of previously created or grabbed pixels. The list, pointed to with CoordinateList.l, will be read in reverse order, facilitating a stack rather than a queue system. This allow this command to be used to correctly restore the pixels in the background which were possibly trashed by other commands. It can also be used simply to render a buffer of coloured pixels, perhaps representing an image, to locations on the bitmap according to the X.w,Y.w, X.q,Y.q or Ptr.l coordinate list. When MParticleMode is set to MSimpleReMapMode or MReMapMode, points will be simple or complex remapped respectively. In MSimpleReMapMode, the buffer points will be simple-remapped and then simply placed in the destination. In MReMapMode, points will be complex-remapped as a combination of buffer pixels and destination pixels. Please refer to MParticleFormat which you should call before performing your particle operations if you want to be sure of the list format.

1.199 mgrabparticlesandplot

```
MGrabParticlesAndPlot CoordinateList.l,NumPoints.l,Buffer.l[,Colour.b]; Grabs ↩
    points to buffer & plots table
```

Does the same as MGrabParticles, at the same time as doing the same as MPlotParticles. As each particle is grabbed from the currently used chunky Bitmap, according to the list of X.w,Y.w, X.q,Y.q or Ptr.l coordinates listed as pointed to by CoordinateList.l, pixels in the currently used Ink or the specified Colour will be plotted to the same coordinates. This facilitates a kind of 'buffered plot' system whereby the background is stored using the grab and then pixels rendered using the plot. Buffer.l points to the buffer memory to which the background pixels will be grabbed and NumPoints.l specifies how many points will be handled. If MParticleMode is set to MSimpleReMapMode or MReMapMode, pixels will be simple or complex remapped respectively when they are plotted. If MReMapMode is chosen, the source colour will be taken from the specified colour or ink. In MSimpleReMapMode the destination pixels will be simple-remapped regardless of specified or currently used colour. Please refer to MParticleFormat which you should call before performing your particle operations if you want to be sure of the list format.

1.200 maddtoparticles

```
MAddToParticles CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add [two sets of]  ↩
    increments to particle list
```

Processes a list of X.w,Y.w, X.q,Y.q or Ptr.l coordinates as pointed to by CoordinateList.l, adding X.w,Y.w, X.q,Y.q or Ptr.l increments to each pair as taken from the list pointed to by IncA.l. The increments list should contain pairs of words, quicks or a mem pointer which are amounts to be added to the X and Y components or the Ptr.l in the coordinate list. The end result is that the particle locations are moved. The increment list could be a set of carefully worked out 'adders', or perhaps a list of precalculated random numbers. This command does not render any particles, it simple moves them. If a second increment list is specified its X.w,Y.w, X.q,Y.q or Ptr.l values will also be added to the particles. Please refer to MParticleFormat which you should call before performing your particle operations to be sure of the list format.

1.201 mwrapparticles

```
MWrapParticles CoordinateList.l,NumPoints.l ; Bring particles in from opposite  ↩
    edge to which they left
```

Checks the particles in the list of X.w,Y.w, X.q,Y.q or Ptr.l coordinates as pointed to by CoordinateList.l to see if they are outside of the bitmap or clip window dimensions, and if so it makes the particle re-enter at the opposite edge to which it left. This feature only works within reason, in that the zone of detection around the edge of the bitmap or clip window is the same width as the bitmap or clip window itself. Particles further away than that will not be properly wrapped. An example is that a pixel may leave the bottom of the bitmap, the wrap will detect this and subtract the height of the bitmap from the Y coordinate, so that the pixel appears to have re-entered at the top. Note also that horizontal wrapping does not work for Ptr.l particle lists, in which case only the highest and lowest memory addresses are considered as the edges. Please refer to MParticleFormat which you should call before performing your particle operations and to be sure of the list format.

1.202 mreboundparticles

```
MReboundParticles CoordinateList.l,NumPoints.l,DirectionList.l,DetectSize.w ;  ↩
    Bounce off edges (NOT Ptr.l!!!)
```

This command will attempt to bounce moving particles off of the edges of the bitmap or clip window. The size of the border of detection is defined with DetectSize.w. This is how close to the edge a pixel must be before its direction will be reversed. Note that pixels need to be at least within the border in order to cause this action, and care should be taken to ensure the border is wide enough to trap even the fastest moving particles. CoordinateList.l points to a list of X.w,Y.w or X.q,Y.q pairs of coordinates and DirectionList.l points to a list of Xmove.w,Ymove.w or xmove.q,ymove.q pairs of movement adders. If the routie detects that a pixel is close enough to the edge of the bitmap or clip window, the corresponding Xmove and Ymove will be negated, which brings about a reversing of direction. Not that there is NO support for Ptr.l particles with rebound because you need seperate X and Y positions in order for the rebound to

work properly, so if you are in Ptr.l mode nothing will happen. Please refer to MParticleFormat which you should call before performing your particle operations to be sure of the list format.

1.203 mprocessor

```
MProcessor ; Returns value 0..6 representing MC68000..MC68060 cpu according to ↩
    exec\AttnFlags
```

Replaces the blitz instruction 'Processor' to provide a function to return a value representing what cpu the host computer has available. Values from 0 to 6 will be returned, representing MC68000 to MC68060 cpu's. Normally most of Mildred will take any value >=4 as meaning that special 040 only routines should be used, where possible. The only difference is that the old 'Processor' instruction did not return values higher than 4, so it was not possible to detect the presence of an 060 cpu. It is recommended that you use 'MProcessor' in place of 'Processor' ↩
    where
possible. Also note that as of v1.36, Mildred will auto-detect the available cpu using MProcessor and will set default values for MCPU and Mc2pCPUmode accordingly. Care should be taken when choosing a cpu that is not the same as the cpu on the host system, although this will not often be required or efficient.

1.204 maddxytoparticles

```
MAddXYToParticles CoordinateList.l,NumPoints.l,XToAdd.w,YToAdd.w ; Add constants ↩
    to all particles
```

Adds constant value XToAdd.w to X.w in the particle list, and constant value YToAdd.w to Y.w in the particle list. The constants are added to all particles. To ignore a component, an adder of 0 is permitted. If either of XToAdd or YToAdd are 0, optimised routines will be used.

1.205 maddxytoparticlesa

```
MAddXYToParticlesA CoordinateList.l,NumPoints.l,ValueToAdd.l ; Add constant to all ↩
     particle pointers
```

Adds a constant value ValueToAdd.l to the Ptr.l components of a particle list. The value will be added to all particles in the list.

1.206 maddxytoparticlesq

```
MAddXYToParticlesQ CoordinateList.l,NumPoints.l,XToAdd.q,YToAdd.q ; Add constants ↩
    to all particles
```

Adds constant value XToAdd.q to X.q in the particle list, and constant value YToAdd.q to Y.q in the particle list. The constants are added to all particles.

To ignore a component, an adder of 0 is permitted. If either XToAdd or YToAdd are 0, optimised routines will be used.

1.207 mparticlemode

MParticleMode [([]Mode.w[])] ; MColourMode, MSimpleReMapMode or MReMapMode - to use in particle plot/draw

Allows you to choose a rendering mode for use in the particle animation commands. Legal parameters are MColourMode, MSimpleReMapMode, or MReMapMode. The default is MColourMode, which will cause things to behave normally. If a remappig mode is chosen, the plot, draw, and grab&plot routines will behave differently. In MReMapMode, the plot routines will take the 'source' colour from the specified colour or the current ink colour, whereas in MSimpleReMapMode it will just remap the destination. The same applies to the MGrabParticlesAndPlot commands. The particle draw commands such as MDrawParticles will output the buffer normally in MColourMode. In MSimpleReMapMode they will simple-remap the buffer data and output it to the destination, in which the data is NOT combined with the destination. In MReMapMode the buffer data will be the source and the destination data the destination for a normal 2-dimensional remap. If MParticleMode is called as a function, it will return the number representing the previously set / current mode.

1.208 mmildredbase

MMildredBase ; Returns long address of the base of Mildred's internal data area

Using this command you can obtain the address in memory at which Mildred's internal data area is stored. The internal data area is built into the library so it will not fluctuate at runtime. Using this command you can pass the base address to your own custom library so that you can access Mildred's data from your custom routines. Note also, library programmers, that it should be possible to obtain this base address by specifying that you want it as part of the !libs macro for a given token in your library header. I'm not sure what you should call the lib, perhapas #MildredLib. In the event that that method doesn't work or is unreliable, MMildredBase can be used to feed the base into your library, which it can store somewhere internally, which may actually be the preferred method. Note also that Mildred's internal data area isn't much of a 'structure' so is liable to possible changes in future versions.

1.209 mdrawingmode

MDrawingMode [()Mode.w[)] ; InvMode/MColourMode/MReMapMode/MSimpleReMapMode to use for drawing (MPlot etc)

The drawing commands such as MPlot, MLine, MCls etc, usually perform their operations in MColourMode. However, using MDrawingMode to specify what mode to use it is possible to render using other methods. Allowed modes are InvMode, MColourMode, MReMapMode and MSimpleReMapMode. In InvMode all drawing operations will invert the destination image. In MColour mode all drawing operations will

just draw normally in the selected colour or the previously specified ink. In MReMapMode the drawing will be performed in such a way that the destination is combined with the currently used or specified colour or ink, using a complex 2-dimensional remap, which will use the currently used Table object. In SimpleReMapMode the destination will be remapped using a 1-dimensional remapping method using the currently used Table object. Note that if you set the drawing mode to do remapping and you try to draw to a stencil or cookie, the mode will be temporarily switched to MColourMode, as it is illogical to be performing remapping to the stencil/cookie.

1.210 mparticleformat

```
MParticleFormat [()Format.b[])] ; Set particle lists/operation format. 0 = X.w,Y.w, ↩
    <0 = X.q,Y.q, >0 = Ptr.l
```

Using this command you can tell the library the format of your particle lists. Specifying a value of 0 indicates that you store your particles as pairs of X.w,Y.w coordinates. If you specify a value less than 0 (perhaps -1) then the system will consider your particles to be stored as pairs of X.q,Y.q coordinates in which each particle consumes 2 longwords (the amount of mem needed to store two 'quick' values). If you specify a value greater than 0 (perhaps 1) then the system will consider your particles to be stored as a single Ptr.l for each pixel, which is an actual memory address. Particles will be grabbed, drawn, plotted to or added to this exact address. You should call MParticleFormat before proceeding to call any other particle commands otherwise the system will default to assuming that you use X.w,Y.w pairs for each particle.

1.211 mpicturedissolveout

```
MPictureDissolveOut PictureBitmapNum.w,Colour.b,WipeToColour.b ;Do picture-based  ↩
   colour dissolve-out of bitmap
```

Performs a picture-based dissolve in order to 'wipe out' a destination bitmap. You first have to 'use' the destination bitmap with MUseBitmap. Then you specify the 'Picture Bitmap' as a parameter to this command which will be the number of a chunky Bitmap containing an 'effect' picture. The routine will search for the location of pixels in the effect picture that are of the specified colour. When it finds one it will draw a pixel at those coordinates in the dest Bitmap to the colour chosen in WipeToColour.b. Note that the picture bitmap and the dest bitmap in this operation have to be the same size. To fully wipe-out a whole picture you should call this command as many times as there are colours in the effect-picture so that all pixels are eventually wiped. The more colours you make use of in the effect-picture the longer the wipe will take. Note that the effect-picture should still be 256-colours even if you only use 64, for example. To create an effect-picture that produces a nice screenwipe you should create a simple greyscale gradient in something like Deluxe Paint and then perform some good image-processing operations on it in something like ImageFX to distort and mangle the gradient to bring variety into the effect. If the destination bitmap starts off blank it will just cause the effect to appear in a new colour rather than removing a graphic.

1.212 mblockunqueue

```
MBlockUnQueue QueueNumber.w[,FirstItem.w,LastItem.w][,BitmapNumber.w];Block- ↩
    UnQueue [range of] objects[&flush]
```

Block-UnQueues the specified Queue to the current chunky Bitmap. If the Bitmap parameter is omitted a 'clearscreen' operation will be performed (to current ink) in all of the areas that have been recorded into the Queue list. If the Bitmap parameter is specified a chunky Bitmap will be used as a background store, from which the areas recorded in the Queue will be blitted to the currently used Bitmap instead of just performing a clearscreen. Once all items have been un-queued, the Queue will be completely flushed. If FirstItem and LastItem are specified, a range of items in the queue will be unqueued. However, in this case the queue will NOT be flushed, and an MFlushQueue will likely be needed at a later stage. The width of items you place on the queue should be a multiple of 16, and their X coordinate should be a multiple of 16. MBlockUnQueue will not check this so it is up to you to be careful in adding items to a queue. Note that if you temporarily select 'MCPU 3' or lower, an 030 routine will be used which does allow you to ignore the alignment of the X coordinates, but still needs the width to be a multiple of 16.

1.213 mwrapxparticles

```
MWrapXParticles CoordinateList.l,NumPoints.l ; Bring particles in from left/right  ↩
    edges (Not Ptr.l)
```

Does the same as MWrapParticles except that only the X.w or X.q component of each particle is wrapped. This means that if a particle goes off the left or right edge of the bitmap or clip window, it will be horizontally wrapped around. Note that there is no routine for Ptr.l particles.

1.214 mwrapyparticles

```
MWrapYParticles CoordinateList.l,NumPoints.l ; Bring particles in from top/bottom  ↩
    edges
```

Does the same as MWrapParticles except that only the Y.w or Y.q component of each particle is wrapped. This means that if a particle goes off the top or bottom edge of the bitmap or clip window, it will be vertically wrapped around. Note that there is no routine for Ptr.l particles.

1.215 maddtoxparticles

```
MAddToXParticles CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add X components  ↩
    of [two sets of] increments
```

Does the same as MAddToParticles except that only the X.w or X.q component of each particle is added to. The coordinate list and increment list(s) should still be pairs of X.w,Y.w or X.q,Y.q, but only the X components of each pair will be added. Note there is no routine for Ptr.l particles.

1.216 maddtoyparticles

```
MAddToYParticles CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add Y components ↩
    of [two sets of] increments
```

Does the same as MAddToParticles except that only the Y.w or Y.q component of each particle is added to. The coordinate list and increment list(s) should still be pairs of X.w,Y.w or X.q,Y.q, but only the Y components of each pair will be added. Note there is no routine for Ptr.l particles.

1.217 mzoom

```
MZoom SrcX.q,SrcY.q,XAdd.q,YAdd.q,DestX.w,DestY.w,OpWidth.w,OpHeight.w,DeRes?.w[, ↩
    SrcBmap.w[,CustomOffsets.l]]
```

Performs a zoom operation on a section of a source bitmap and places it into the destination bitmap at a different size. SrcX.q and SrcY.q specify the coordinate of the top left corner within the source image at which to start reading. Note that these are .quick values so fractional starting positions are possible. XAdd.q and YAdd.q are .quick values which are incremented to the SrcX.q and SrcY.q variables for each pixel in the operation to move through the source image at a specific rate. Note that there are optimised routines for when either or both adders have no fractional part (ie are whole numbers), which is particularly faster when XAdd.q has no fractional part, and will also be faster if you can live with an OpWidth that is a multilpe of 4, or better even a multilpe of 16. XAdd and YAdd of 1.0 causes the equivelent of a straight 'scroll' of the image, although slower than normal scrolls. DestX.w and DestY.w are the coordinates of the top left corner of the area to be zoomed in the destination. These are integer values and correspond to a precice pixel coordinate. OpWidth.w is the width of the operation in whole pixels and OpHeight.w is the height of the operation in whole pixels. This defines the area in the destination that will be rendered to. You can optionally specify a source bitmap to use, otherwise the source and destination will be the same bitmap. If you specify CustomOffsets.l a special zoom will be performed which reads values from a list in order to perform different zoom factors for each row. The list should contain groups of OpWidth.w,DestXOffset.w,SrcXOffset.q, XAddOffset.q,YAddOffset.q. Each set of 5 variables represents values to use for a single row of the zoom. OpWidth.w is the absolute width of the operation in whole pixels, and represents the number of pixels to be zoomed on a given row. DestXOffset.w is a relative offset in the destination, relative to the DestX coordinate of the previous row. An offset of 0 on every line causes a normal zoom whereas an offset of 1 on every line would shear the area at 45 degrees. SrcXOffset.q is a relative value to add to the X coordinate of the start of the row in the source, relative to the previous row. For a straight verticle edge you would have an offset of 0 for each row. Note that SrcXOffset.q is a .quick value so can be a fractional amount. XAddOffset.q is a relative value to add to the XAdd.q variable for the given row, relative to the previous value of XAdd.q. A XAddOffset of 0 produces an evenly distributed zoom whereas a XAddOffset of something like 0.001 may cause the zoom effect to stretch horizontally as the processing travels from left to right on a row. YAddOffset.q is a relative value to add to the YAdd.q variable for the given row, relative to the previous value of YAdd.q. A YAddOffset of 0 produces an evenly distributed zoom whereas a YAddOffset of something like 0.001 may cause the zoom effect to stretch vertically as the processing travels from top to bottom. The DeRes.w

parameter should be used to tell Mildred what type of zoom to perform. Normally this should be set to False or 0, to indicate that a normal zoom will occur. If you specify DeRes.w as True or <>0 a de-resolution effect will occur. De-resolution means a decrease in the resolution of the image, but without a change to its overall size. OpWidth and OpHeight will define the size of the image to be affected, and the zoom effect dictates the resolution. What would originally have been a zoomed 'pixel' will now become an solid block of colour on the image, of the same size but covering several pixels rather than representing just one. Note that there are optimised routines for when the OpWidth is a multiple of 4, 16, when the adders are integers and when the Y adder is 1.0 (for a much faster Y zoom/deres). It is also possible to use DeRes.w=True in combination with CustomOffsets.l to provide different levels of de-resolution on each line.

1.218 mzoomshape

```
MZoomShape SrcX.q,SrcY.q,XAdd.q,YAdd.q,DestX.w,DestY.w,OpWidth.w,OpHeight.w,DeRes ↩
    ?.w[,SrcShap.w[,CustOffs.l]]
```

Does the same as MZoom except from one shape to another.

1.219 mzoombitmaptoshape

```
MZoomBitmapToShape SrcX.q,SrcY.q,XAdd.q,YAdd.q,DstX.w,DstY.w,OpWid.w,OpHeight.w, ↩
    DeRes?.w[,SrcBmap.w[,CustOffs.l]]
```

Does the same as MZoom except with a bitmap as the source and a shape as the destination.

1.220 mzoomshapetobitmap

```
MZoomShapeToBitmap SrcX.q,SrcY.q,XAdd.q,YAdd.q,DstX.w,DstY.w,OpWidth.w,OpHeight.w, ↩
    DeRes?.w[,SrcShap.w[,CustOffs.l]]
```

Does the same as MZoom except with a shape as the source and a bitmap as the destination.

1.221 maddmode

```
MAddMode ; Returns value 7 which represents 'add' mode in the blit modes
```

Returns value 7 which is the new 'add' mode. This affects all drawing commands, blits, scrolls and particles which allow you to select a rendering mode. In Add Mode, the pixels of the source and destination images, or alternatively the pixels of the destination and ink colour, are simply added together and the result placed in the destination. To cause a 'subtraction' effect you can simply add 256-Value in order to achieve the same effect. Add mode can be used for

simple software colour cycling and for faster shade-bob type effects, for example.