Mildred

COLLABORATORS

	TITLE : Mildred	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Mildred

1.1 Mildred v1.38

```
                Here is the documentation for Mildred v1.38.

Mildred v1.38 and any earlier versions are Copyright (c) 1998-1999 Paul West and
Pagan Software.


                 Information

                 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+

                 M040c2pUsage
                 Status.b ; On/Off - Availability of 040 c2p. Overrides  ↩
                    Mc2pCPUmode

                 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

                 Mc2pRowLacing
```

```
State.b ; Toggle row-lacing in c2p On/Off. NonZero=On

Mc2pColumnLacing
State.b ; Toggle column-lacing in c2p On/Off

Mc2pRowToggle
; Toggle c2p row lacing between Even/Odd rows

Mc2pColumnToggle
; Toggle c2p column lacing between Even/Odd columns

Mc2pToggleSingle
; Toggle c2p lacing for single-buffered display

Mc2pToggleDouble
Buf.b ; 0 or 1. Toggle c2p lacing for double-buffered display

Mc2pToggleTriple
Buf.b ; 0, 1 or 2. Toggle c2p lacing for triple-buffered display

Mc2pToggle
Buffers.b,Buf.b ; 1, 2 or 3, and 0, 1 or 2. Toggle c2p lacing.

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)

MFreec2pWindows
[Firstc2pWindow.w,Lastc2pWindow.w] ; Free/delete all/range of  ↩
   c2pwindows

MFreec2pWindow
 Free/delete a pre-existing c2pWindow

MUsec2pWindows
Mainc2pWindowNum.w[,Secondc2pWindowNum.w[,Thirdc2pWindowNum.w]] ; ↩
     Current to use

MUsec2pWindow
c2pWindowNumber.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

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

                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

                MFreeShapes
                [FirstShape.w,LastShape.w] ; Free/delete all/range of Shapes

                MFreeShape
                ShapeNumber.w ; Free/delete a pre-existing Shape

                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

MMidHandle
ShapeNumber.w ; Set handle to middle of Shape

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

MUseShape
ShapeNumber.w ; Current Shape to use

MUsedShape
 ; Returns currently used Shape

MMakeCookies
[FirstShape.w,LastShape.w] ; Make cookies for all/range of shapes

MMakeCookie
ShapeNumber.w ; Make a cookie for a shape

MFreeCookies
[FirstShape.w,LastShape.w] ; Free all/range of cookies

MFreeCookie
ShapeNumer.w ; Free's the Shape's cookie

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

MCopyHandle
SourceShapeNumber.w,DestShapeNumber.w ; Copy a shape's handle to  ↩
    another 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

MFreeBitmaps
[FirstBitmap.w,LastBitmap.w] ; Free/delete all/range of Bitmaps

MFreeBitmap
BitmapNumber.w ; Free/delete a pre-existing Bitmap

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

MBitmapMidOrigin
BitmapNumber.w ; Set origin to middle of Bitmap

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

MUseBitmap
BitmapNumber.w ; Current Bitmap to use

MUsedBitmap
; Returns currently used Bitmap

MMakeStencils
[FirstBitmap.w,LastBitmap.w] ; Make stencils for all/range of bitmaps

MMakeStencil
BitmapNumber.w ; Make a stencil for a bitmap

MFreeStencils
[FirstBitmap.w,LastBitmap.w] ; Free all/range of stencils

MFreeStencil
BitmapNumber.w ; Free's the Bitmap's stencil

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

MCopyOrigin
SourceBitmapNumber.w,DestBitmapNumber.w ; Copy a bitmap's origin to another 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] ; Copy  ↩
    graphic

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

MScrollStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Copy  ↩
    stencil to stencil only

MScrollCookie
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy  ↩
    cookie to cookie only
```

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

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

MScrollStencilToCookie
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Copy ↩
    stencil to cookie

MScrollCookieToStencil
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy ↩
    cookie to stencil

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] ; Copy  ↩
    bm 2 bm and st 2 st

MSScrollShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy sh ↩
     2 sh and ck 2 ck

MSScrollBitmapToShape
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; Copy  ↩
    bm 2 sh and st 2 ck

MSScrollShapeToBitmap
X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy sh ↩
     2 bm and ck 2 st

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


                MInk
                Colour.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) ; Return the colour of a single pixel in a bitmap

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

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

MPointCookie
(Xpos.w,Ypos.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[,BitmapNumber.w] ; UnQueue the queued objects and  ↩
                   flush the queue

                MUnQueueRange
                QueueNumber.w,FirstItem.w,LastItem.w[,BitmapNumber.w] ; UnQueue a ↩
                    range of queued objects

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

                MFreeQueues
                [FirstQueue.w,LastQueue.w] ; Free/delete all/range of Queues

                MFreeQueue
                QueueNumber.w ; Free/delete a pre-existing Queue

                MAddrQueue
                (QueueNumber.w} ; Returns address of Queue structure

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

                MUseQueue
                QueueNumber.w ; Current to use

                MUsedQueue
                 ; Returns currently used Queue

blits


                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)

                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

                MFreeTables
```

```
                [FirstTable.w,LastTable.w] ; Free/delete all/range of Tables

                MFreeTable
                TableNumber.w ; Free/delete a pre-existing Table

                MAddrTable
                (TableNumber.w} ; Returns address of Table structure

                MFlushTable
                TableNumber.w ; Empties the table to contain no items

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

                MUseTable
                TableNumber.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

                MReMapUsingShape
                RemapTable.l[,SourceShapeNum.w[,DestBitmapNum.w]] ;Merge shape to ↩
                     bitmap using 2xWay Table

                MReMapShapeUsingShape
                RemapTable.l[,SourceShapeNum.w[,DestShapeNum.w]] ;Merge shape to  ↩
                    shape using 2xWay Table

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

particles
```

```
MParticleMode
Mode.w ; MColourMode, MSimpleReMapMode or MReMapMode - to use in particle plot/draw

MPlotParticles
CoordinateList.l,NumPoints.l[,Colour.b] ; Plot lots of points from an X.w,Y.w table of coords

MGrabParticles
CoordinateList.l,NumPoints.l,Buffer.l ; Grab lots of points from X.w,Y.w table, into buffer mem

MDrawParticles
CoordinateList.l,NumPoints.l,Buffer.l ; Draw lots of previously grabbed points, using X.w,Y.w's

MGrabParticlesAndPlot
CoordinateList.l,NumPoints.l,Buffer.l[,Colour.b]; Grabs points X.w,Y.w to buffer & plots

MAddToParticles
CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add X.w,Y.w to X.w,Y.w items in particle list

MReboundParticles
CoordinateList.l,NumPoints.l,DirectionList.l,DetectSize.w ; Bounce particles off edges

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

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

MPlotParticlesA
AddressList.l,NumPoints.l[,Colour.b] ; Plot lots of points from an Ptr.l table of coords

MGrabParticlesA
AddressList.l,NumPoints.l,Buffer.l ; Grab lots of points from Ptr.l table, into buffer mem

MDrawParticlesA
AddressList.l,NumPoints.l,Buffer.l ; Draw lots of previously grabbed points, using Ptr.l's

MGrabParticlesAndPlotA
AddressList.l,NumPoints.l,Buffer.l[,Colour.b] ; Grabs points Ptr.l to buffer & plots

MAddToParticlesA
AddressList.l,NumPoints.l,IncA.l[,IncB.l] ; Add Ptr.l to Ptr.l items in particle list
```

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

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

                MPlotParticlesQ
                CoordinateList.l,NumPoints.l[,Colour.b] ; Plot lots of points  ↩
                    from an X.q,Y.q table of coords

                MGrabParticlesQ
                CoordinateList.l,NumPoints.l,Buffer.l ; Grab lots of points from  ↩
                    X.q,Y.q table, to buffer mem

                MDrawParticlesQ
                CoordinateList.l,NumPoints.l,Buffer.l ; Draw previously grabbed  ↩
                    points, using X.q,Y.q's

                MGrabParticlesAndPlotQ
                CoordinateList.l,NumPoints.l,Buffer.l[,Colour.b];Grabs points X.q ↩
                    ,Y.q to buffer & plots

                MAddToParticlesQ
                CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add X.q,Y.q to X.q ↩
                    ,Y.q items in particle list

                MReboundParticlesQ
                CoordinateList.l,NumPoints.l,DiectionList.l,DetectSize.w ; Bounce ↩
                     particles off edges

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

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

 others


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

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.
The library itself is also Copyright (c) 1998-1999 Pagan Software.
```

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.
```

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 - Addred 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] !!!
```

1.4 mc2prowlacing

```
Mc2pRowLacing State.b ; Toggle row-lacing in c2p On/Off. NonZero=On
```

Switches the row-interlacing feature of the c2p sytem On or Off. Using this command will affect the information held in any c2pWindow objects created afterwards, and will cause the c2p and other elements of the system to behave in the necessary way for supporting row-interlaced output. Using this command also resets the even/odd frame to even (0).

1.5 mc2pcolumnlacing

```
Mc2pColumnLacing State.b ; Toggle column-lacing in c2p On/Off
```

Switches the column-interlacing feature of the c2p system On or Off. A column is 32 pixels wide. Using this will affect the information held in any c2pWindow objects created afterwards, and will cause the c2p and other elements of the system to behave in a way necessary for supporting column-interlaced output. Using this command also resets the even/odd frame to even (0).

1.6 mc2prowtoggle

```
Mc2pRowToggle  ; Toggle c2p row lacing between Even/Odd rows
```

Toggles the current frame for row-interlacing between even and odd, and vice versa ↩
,
in order to cause the row-interlacing effect with successive calls to the chunky-to-planar converter.

1.7 mc2pcolumntoggle

```
Mc2pColumnToggle  ; Toggle c2p column lacing between Even/Odd columns
```

Toggles the current frame for column-interlacing between even and odd, and vice versa, in order to cause the column-interlacing effect with successive calls to the chunky-to-planar converter.

1.8 mc2ptogglesingle

```
Mc2pToggleSingle  ; Toggle c2p lacing for single-buffered display
```

Performs both row and column interlacing toggles in the c2p system (if applicable) in a manner that is designed for use with single-buffered displays. Not that it may take 2-4 conversions in order to produce a full display.

1.9 mc2ptoggledouble

```
Mc2pToggleDouble Buf.b ; 0 or 1. Toggle c2p lacing for double-buffered display
```

Performs both row and column interlacing toggles in the c2p system (if applicable) in a manner that is designed for use with double-buffered displays. Note that it may take 2-4 conversions in order to produce a full display.

1.10 mc2ptoggletriple

```
Mc2pToggleTriple Buf.b ; 0, 1 or 2. Toggle c2p lacing for triple-buffered display
```

Performs both row and column interlacing toggles in the c2p system (if applicable) in a manner that is designed for use with triple-buffered displays. Note that it may take 2-6 conversions in order to produce a full display.

1.11 mc2ptoggle

```
Mc2pToggle Buffers.b,Buf.b ; 1, 2 or 3, and 0, 1 or 2. Toggle c2p lacing.
```

Performs row and column interlacing toggles (if applicable) in the c2p system in a manner that is designed for use with single, double, or triple-buffered displays, in which you must specify the current frame number (0..Buffers-1). Normally you should toggle your buffers with something like Buf=1-Buf, or a slighly more complicated cycling of the current number for triple-buffered displays. This command effectively does the same as Mc2pToggleSingle, Mc2pToggleDouble or Mc2pToggleTriple, although you do have more manual control over frame number (be careful with misuse of it or you might get wierd results).

1.12 m040c2pusage

```
M040c2pUsage Status.b ; On/Off - Availability of 040 c2p. Overrides Mc2pCPUmode
```

The c2p system has routines for 030 or lower, and 040 or higher cpu's. In order to completely shut out the 040 routine in all circumstances, this command should be used. Even if you set the c2pCPUmode to use the 040 routine and have enabled 040 usage, the c2p system will use the 030 routine. You do not have to worry about the 040 c2p routine running on a lower cpu as it will not crash. It will simply be less efficient.

1.13 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. If you set the mode to use the 040+ routine, M040c2pUsage can be used to toggle between 040 and 030 modes. 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.

1.14 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.

1.15 mc2pwindowwidth

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

Returns the operation width as defined in a previously created c2pWindow object.

1.16 mc2pwindowheight

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

Returns the operation height as defined in a previously created c2pWindow object.

1.17 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.

1.18 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.

1.19 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.

1.20 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.

1.21 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.

1.22 minitshape

```
MInitShape [()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.

1.23 mshape

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

Exactly the same as MInitShape (see above).

1.24 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.

1.25 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.

1.26 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).

1.27 mfreec2pwindows

```
MFreec2pWindows [Firstc2pWindow.w,Lastc2pWindow.w] ; Free/delete all/range of ↩
    c2pwindows
```

Allows you to free 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 or otherwise will free a range of objects (and will error-check that they exist first).

1.28 mfreec2pwindow

```
MFreec2pWindow  Free/delete a pre-existing c2pWindow
```

Free's a specific previously-created c2pWindow. If the c2pWindow doesn't exist it will generate an error.

1.29 mfreeshapes

```
MFreeShapes [FirstShape.w,LastShape.w] ; Free/delete all/range of Shapes
```

Allows you to free 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, but if you specify a range of objects to free it will check that each one exists first and if not will generate an error.

1.30 mfreeshape

```
MFreeShape ShapeNumber.w ; Free/delete a pre-existing Shape
```

Free's a specific previously-created chunky Shape. If the Shape doesn't exist it will generate an error. The Shape's definition and grapics-data (and cookie) will be freed.

1.31 mfreebitmaps

```
MFreeBitmaps [FirstBitmap.w,LastBitmap.w] ; Free/delete all/range of Bitmaps
```

Allows you to free 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, but if you specify a range of objects to free it will check that each one exists first and if not will generate an error.

1.32 mfreebitmap

```
MFreeBitmap BitmapNumber.w ; Free/delete a pre-existing Bitmap
```

Free's a specific previously-created chunky Bitmap. If the Bitmap doesn't exist it will generate an error. The Bitmap's definition and graphics-data (and stencil) will be freed.

1.33 mshapewidth

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

Returns the width of a previously created chunky Shape object.

1.34 mbitmapwidth

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

Returns the width of a previously created chunky Bitmap object.

1.35 mshapeheight

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

Returns the height of a previously created chunky Shape object.

1.36 mbitmapheight

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

Returns the height of a previously created chunky Bitmap object.

1.37 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.

1.38 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.

1.39 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.

1.40 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.

1.41 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.

1.42 mmidhandle

```
MMidHandle ShapeNumber.w ; Set handle to middle of Shape
```

Sets the handle for a previously-created chunky Shape object to the center of the Shape.

1.43 mbitmapmidorigin

```
MBitmapMidOrigin BitmapNumber.w ; Set origin to middle of Bitmap
```

Sets the handle for a previously-created chunky Bitmap object to the center of the Bitmap.

1.44 musec2pwindows

```
MUsec2pWindows 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.
```

1.45 musec2pwindow

```
MUsec2pWindow c2pWindowNumber.w ; Current to use

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

1.46 museshapes

```
MUseShapes 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.
```

1.47 museshape

```
MUseShape ShapeNumber.w ; Current Shape to use

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

1.48 musebitmaps

```
MUseBitmaps 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.
```

1.49 musebitmap

```
MUseBitmap BitmapNumber.w ; Current Bitmap to use
```

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

1.50 musedc2pwindow

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

Returns the number of the currently used main c2pWindow object.

1.51 musedshape

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

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

1.52 musedbitmap

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

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

1.53 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. Ensure that you have allocated at least 16 bytes more memory than you need as the Shape will modify the base address in order to facilitate proper block-handling routines on higher cpu's. Width should be a multiple of 4. The base-address is modified with the expression: (base=base+16) and $FFFFFFF0. 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.

1.54 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. Ensure that you have allocated at least 16 bytes more memory than you need as the Bitmap will modify the base address in order to facilitate proper block-handling routines on higiher cpu's. Width should be a multiple of 4. The base-address is modified with the expression: (base=base+16) and $FFFFFFF0. 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.

1.55 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.

1.56 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.

1.57 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.

1.58 mmakecookies

```
MMakeCookies [FirstShape.w,LastShape.w] ; Make cookies for all/range of shapes
```

Initialises (if necessary) and generates cookies for all or a range of chunky Shape objects. 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 chunnky Shapes will have cookies generated. The area of a Shape used to make a cookie can be cropped using the Shape's clip window.

1.59 mmakecookie

```
MMakeCookie ShapeNumber.w ; Make a cookie for a shape
```

Initialises (if necessary) and generates a cookie for a specific chunky Shape object. 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. The area of the Shape used to make a cookie can be cropped using the Shape's clip window.

1.60 mmakestencils

```
MMakeStencils [FirstBitmap.w,LastBitmap.w] ; Make stencils for all/range of ↩
    bitmaps
```

Initialises (if necessary) and generates cookies for all or a range of chunky Bitmap objects. If the cookie 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 the Bitmap used to make a stencil can be cropped using the Bitmap's clip window.

1.61 mmakestencil

```
MMakeStencil BitmapNumber.w ; Make a stencil for a bitmap
```

Initialises (if necessary) and generates a stencil for a specific chunky Bitmap object. 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. The area of the Bitmap used to make a stencil can be cropped using the Bitmap's clip window.

1.62 mfreecookies

```
MFreeCookies [FirstShape.w,LastShape.w] ; Free all/range of cookies
```

Free's 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.

1.63 mfreecookie

```
MFreeCookie ShapeNumer.w ; Free's the Shape's cookie
```

Free's a specific previously-created cookie in a previously-created chunky Shape object. Only the cookie is freed.

1.64 mfreestencils

```
MFreeStencils [FirstBitmap.w,LastBitmap.w] ; Free all/range of stencils
```

Free's 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.

1.65 mfreestencil

```
MFreeStencil BitmapNumber.w ; Free's the Bitmap's stencil
```

Free's a specific previously-created stencil in a previously-created chunky Bitmap object. Only the stencil is freed.

1.66 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.

1.67 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.

1.68 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.

1.69 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.

1.70 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.

1.71 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.

1.72 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.

1.73 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.

1.74 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.

1.75 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.

1.76 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.

1.77 mcopyhandle

```
MCopyHandle SourceShapeNumber.w,DestShapeNumber.w ; Copy a shape's handle to ↩
    another shape
```

Copies the handle from one previously-created chunky Shape to another.

1.78 mcopyorigin

```
MCopyOrigin SourceBitmapNumber.w,DestBitmapNumber.w ; Copy a bitmap's origin to ↩
    another bitmap
```

Copies the handle from one previously-created chunky Bitmap to another.

1.79 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.

1.80 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.

1.81 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.

1.82 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.

1.83 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.

1.84 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.

1.85 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.

1.86 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.

1.87 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.

1.88 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.

1.89 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.

1.90 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.

1.91 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.

1.92 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.

1.93 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.

1.94 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.

1.95 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.

1.96 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.

1.97 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.

1.98 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.

1.99 mscroll

```
MScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; 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.

1.100 mscrollshape

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

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

1.101 mscrollstencil

```
MScrollStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;
```

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.

1.102 mscrollcookie

```
MScrollCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy ↩
    cookie to cookie only
```

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.

1.103 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.

1.104 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.

1.105 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.

1.106 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.

1.107 mscrollbitmaptoshape

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

1.108 mscrollshapetobitmap

```
MScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    Copy shape to 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.

1.109 mscrollstenciltocookie

```
MScrollStencilToCookie X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ;  ↩
    Copy stencil to cookie
```

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.

1.110 mscrollcookietostencil

```
MScrollCookieToStencil X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ;  ↩
    Copy cookie to stencil
```

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.

1.111 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.

1.112 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.

1.113 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.

1.114 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.

1.115 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.

1.116 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.

1.117 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.

1.118 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.

1.119 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.

1.120 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.

1.121 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.

1.122 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.

1.123 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.

1.124 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.

1.125 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.

1.126 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.

1.127 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.
```

1.128 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.
```

1.129 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.
```

1.130 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.
```

1.131 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.
```

1.132 mpoint

```
MPoint (Xpos.w,Ypos.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.

1.133 mpointshape

```
MPointShape (Xpos.w,Ypos.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.

1.134 mpointstencil

```
MPointStencil (Xpos.w,Ypos.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 chunky Bitmap is transparent or not, based on the Bitmap's stencil.

1.135 mpointcookie

```
MPointCookie (Xpos.w,Ypos.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 chunky Shape is transparent or not, based on the Shape's cookie.

1.136 msscroll

```
MSScroll X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; 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.

1.137 msscrollshape

```
MSScrollShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; Copy sh 2 ↩
    sh and ck 2 ck
```

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.

1.138 msscrollbitmaptoshape

```
MSScrollBitmapToShape X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceBitmapNum.w] ; ↩
    Copy bm 2 sh and st 2 ck
```

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.

1.139 msscrollshapetobitmap

```
MSScrollShapeToBitmap X1.w,Y1.w,Width.w,Height.w,X2.w,Y2.w[,SourceShapeNum.w] ; ↩
    Copy sh 2 bm and ck 2 st
```

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.

1.140 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.

1.141 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.

1.142 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.

1.143 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.

1.144 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.

1.145 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.

1.146 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.

1.147 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.

1.148 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'.

1.149 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.

1.150 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 MBitmapsUse, 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.

1.151 mmaskscrollmode

```
MMaskScrollMode Mode.w ; CookieMode/0 or EraseMode/1 or InvMode/2 or SolidMode/3
```

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.

1.152 mblitmode

```
MBlitMode Mode.w ; CookieMode/0 or EraseMode/1 or InvMode/2 or SolidMode/3
```

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.

1.153 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.

1.154 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.

1.155 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.

1.156 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.

1.157 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.

1.158 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.

1.159 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.

1.160 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.

1.161 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.

1.162 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.

1.163 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.

1.164 mfreequeues

```
MFreeQueues [FirstQueue.w,LastQueue.w] ; Free/delete all/range of Queues
```

Frees all or a range of previously-created queues. If the parameters are omitted it will free all created chunky Queue objects, otherwise just the range you specify. 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.

1.165 mfreequeue

```
MFreeQueue QueueNumber.w ; Free/delete a pre-existing Queue
```

Frees a specific chunky Queue object. If the chunky Queue doesn't exist it will generate an error. The Queue will be 'killed' and the memory for the Queue's items (if any) will be freed, making the Queue empty.

1.166 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.

1.167 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.

1.168 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.

1.169 mqblitmode

```
MQBlitMode Mode.w ; CookieMode/0 or EraseMode/1 or InvMode/2 or SolidMode/3
```

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.

1.170 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.

1.171 musequeues

```
MUseQueues 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.

1.172 musequeue

```
MUseQueue QueueNumber.w ; Current to use
```

Selects a specific chunky Queue object to use as current. This current Queue will be assumed later so that you don't have to keep specifying which chunky Queue object to use.

1.173 musedqueue

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

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

1.174 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.

1.175 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.

1.176 munqueue

```
MUnQueue QueueNumber.w[,BitmapNumber.w] ; UnQueue the queued objects and flush the ↩
     queue
```

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

is omitted a 'clearscreen' operation will be performed (to Colour 0) 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.

1.177 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.

1.178 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.

1.179 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.

1.180 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.

1.181 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.

1.182 msblitmode

```
MSBlitMode Mode.w ; CookieMode/0 or EraseMode/1 or InvMode/2 or SolidMode/3
```

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.

1.183 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.

1.184 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.

1.185 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.

1.186 mqsblitmode

```
MQSBlitMode Mode.w ; CookieMode/0 or EraseMode/1 or InvMode/2 or SolidMode/3
```

Does the same as MSBlitMode except with reference to the QS-type blits.

1.187 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.

1.188 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.

1.189 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.

1.190 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.

1.191 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.

1.192 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.

1.193 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.

1.194 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.

1.195 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.

1.196 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.

1.197 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.

1.198 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.

1.199 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.

1.200 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.

1.201 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, although bear in mind that it uses the following expression to calculate where the cookie should begin: Base=((Memory+16) and $FFFFFFF0). This is to provide proper support for block and tile routies on 040+ cpu's so be careful that you have reserved at least 16 bytes more memory than is needed to hold the entire cookie.

1.202 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.

1.203 munqueuerange

```
MUnQueueRange QueueNumber.w,FirstItem.w,LastItem.w[,BitmapNumber.w] ; UnQueue a ↩
    range of queued objects
```

Does the same as MUnQueue except that it allows you specify a range of items in the queue. Items are placed in the queue in the order you design and will be numbered from 0 upwards. Once the range of items (which may be only one item) has been un-queued, perhaps with a clearscreen on the region or a blit from a store Bitmap, the specified chunky Queue is NOT flushed so at some point it may become necessary to do an MFlushQueue.

1.204 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.

1.205 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.

1.206 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).

1.207 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.

1.208 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.

1.209 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.

1.210 mremapusingshape

```
MReMapUsingShape RemapTable.l[,SourceShapeNum.w[,DestBitmapNum.w]] ;Merge shape to ↩
     bitmap using 2xWay Table
```

Performs a remapping operation using a previously calculated colour-lookup table. If not specified, the currently used chunky Bitmap object will be used as the destination for the remapping, and if a shape is not specified it will use the currently used chunky Shape object. Pixels are read from the destination Bitmap and from the source Shape and their values are used as an offset into the table (RemapTable.l is an address pointing to the base of your table). Then a byte value, representing what colour the pixel should be changed to, is taken from the appropriate position in the table and output to the destination bitmap. You may wish to use a call to MQDummy to add the affected area to the queue, otherwise multiple remaps to the same area will cause a possibly wrong effect. The width and height of the remap operation are equal to the width and height of the chunky Shape object being used, and the coordinates in the Bitmap at which the remapped area will be rendered are taken from the Shape's handle, which you should set with MHandle, and are also dependent on any possible wrapping of the Bitmap's origin. It does not matter if ShapeWrap is on or not. Normally the operation will output to coordinates 0,0 in the Bitmap, unless you have altered the Shape's handle, and be careful with negative handles or ones which position the operation outside of the Bitmap into unreserved memory because the runtime errorchecking does not check that the operation fits. Using MReMapUsingShape it is possible to merge, mix, add and negate the image, as well as performing various effects such as lighting, smoke, fire, mist/fog, etc, and the table (which should be 256x256 bytes) allows all 256 colours of source and destination to be combined. Offsets in the remapping table should be calculated using the colour from the destination bitmap in the low byte, and the colour from the source shape in the high byte, producing a number from 0..65535. The colour that you would like this combination of colours to produce should then be written to the table at that position in the table.

1.211 mremapshapeusingshape

```
MReMapShapeUsingShape RemapTable.l[,SourceShapeNum.w[,DestShapeNum.w]] ;Merge ↩
    shape to shape using 2xWay Table
```

Does the same as MReMapUsingShape, except that a chunky Shape will be used as the output rather than a chunky Bitmap. If DestShapeNum.w is not specified, the /second/ currently used Shape will be assumed. The first used chunky Shape will be assumed if SourceShapeNum is also omitted.

1.212 mink

```
MInk Colour.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.

1.213 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.

1.214 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.

1.215 mfreetables

```
MFreeTables [FirstTable.w,LastTable.w] ; Free/delete all/range of Tables
```

Frees all of the Table objects that have been created, or a range of objects ranging from FirstTable to LastTable inclusive. 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.

1.216 mfreetable

```
MFreeTable TableNumber.w ; Free/delete a pre-existing Table
```

Frees a single Table object, as specified with TableNumber. If the object does not exist, the runtime errorchecking routine will report an error. Once freed the Table will be effectively dead and free of actual data.

1.217 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.

1.218 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.

1.219 mflushtable

```
MFlushTable TableNumber.w ; Empties the table to contain no items
```

Performs a similar function to MFlushQueue except that it performs it on a Table object. This does not erase any data in the table, it simply resets the internal pointer that points to the 'current' item so that it is pointing to the start of the table. At this time, this command doesn't really relate with any other part of Mildred.

1.220 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.

1.221 musetables

```
MUseTables 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.

1.222 musetable

```
MUseTable TableNumber.w ; Current to use
```

Selects a single Table objet to use as current. This is the main Table that will be used as in MUseTables. Once selected, other routines can assume to use this Table in their operations.

1.223 musedtable

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

Returns the number of the currently used main Table object.

1.224 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.

1.225 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.

1.226 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.

1.227 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.

1.228 mplotparticles

```
MPlotParticles CoordinateList.l,NumPoints.l[,Colour.b] ; Plot lots of points from ↩
    an X.w,Y.w table of coords
```

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 should contain X.w,Y.w pairs. 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.

1.229 mgrabparticles

```
MGrabParticles CoordinateList.l,NumPoints.l,Buffer.l ; Grab lots of points from X. ↩
    w,Y.w 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. 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.

1.230 mdrawparticles

```
MDrawParticles CoordinateList.l,NumPoints.l,Buffer.l ; Draw lots of previously ↩
    grabbed points, using X.w,Y.w's
```

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 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.

1.231 mgrabparticlesandplot

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

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 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.

1.232 maddtoparticles

```
MAddToParticles CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add X.w,Y.w to X.w, ↩
    Y.w items in particle list
```

Processes a list of X.w,Y.w coordinates as pointed to by CoordinateList.l, adding X.w,Y.w increments to each pair as taken from the list pointed to by IncA.l. The increments list should contain pairs of words which are amounts to be added to the X and Y components 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 values will also be added to the particles.

1.233 mplotparticlesa

```
MPlotParticlesA AddressList.l,NumPoints.l[,Colour.b] ; Plot lots of points from an ↩
     Ptr.l table of coords
```

Does the same as MPlotParticles, except that the items on the list are stored as actual memory addresses. Each address (a longword) is the actual address in memory at which a pixel is to be plotted. Many animation effects are possible

using addresses rather than coordinates, and the plotting of pixels is much faster due to the time-consuming calculations being avoided in working out where in memory to output to. There is only one longword per item, which represents the X and Y coordinates combined.

1.234 mgrabparticlesa

```
MGrabParticlesA AddressList.l,NumPoints.l,Buffer.l ; Grab lots of points from Ptr. ↩
    l table, into buffer mem
```

Does the same as MGrabParticles, except that the items on the list are stored as actual memory addresses. Each address (a longword) is the actual address in memory from which a pixel is to be plotted. Many animation effects are posible using addresses rather than coordinates, as the grabbing of pixels is much faster due to the time-consuming calculations being avoided in working out where in memory to grab from. There is only one longword per item, which represents the X and Y coordinates combined. Particles are grabbed to the memory buffer pointed to by Buffer.l.

1.235 mdrawparticlesa

```
MDrawParticlesA AddressList.l,NumPoints.l,Buffer.l ; Draw lots of previously  ↩
    grabbed points, using Ptr.l's
```

Does the same as MDrawParticles, except that the items on the list are stored as actual memory addresses. Each address (a longword) is the actual address in memory to which pixels will be drawn. Many animation effects are possible due to the time-consuming calculations being avoided in working out where in memory to grab from. There is only one longword per item, which represents the X and Y coordinates combined. Particles are drawn from the memory buffer pointed to by Buffer.l, and the whole lot is done in reverse order in order to facilitate a stack system.

1.236 mgrabparticlesandplota

```
MGrabParticlesAndPlotA AddressList.l,NumPoints.l,Buffer.l[,Colour.b] ; Grabs  ↩
    points Ptr.l to buffer & plots
```

Does the same as MGrabParticlesAndPlot, except that the items on the list are stored as actual memory addresses. Each address (a longword) is the actual address in memory at which particles will be grabbed and plotted. Many animation effects are possible due to the time consuming calculations being avoided in working out where in memory to grab from or plot to. There is only one longword per item, which represents the X and Y coordinates combined. Particles are grabbed to the memory buffer pointed to by Buffer.l.

1.237 maddtoparticlesa

```
MAddToParticlesA AddressList.l,NumPoints.l,IncA.l[,IncB.l] ; Add Ptr.l to Ptr.l ↩
    items in particle list
```

Does the same as MAddToParticles, except using a list of longword pointers. The address list is the list of longword pointers representing coordinates, and the list pointed to by Inca.l is a list of longword values to add to those pointers, which could be values representing X and Y combined coordinates to add/subtract. If a second increment list is specified then a second list of Ptr.l values will be added to the particles.

1.238 mplotparticlesq

```
MPlotParticlesQ CoordinateList.l,NumPoints.l[,Colour.b] ; Plot lots of points from ↩
     an X.q,Y.q table of coords
```

Does the same as MPlotParticles except that the list data are pairs of X.q,Y.q. In this respect, a .q quick are longwords each, causing two longwords to be needed for each pixel. The top 16 bits of each component will be used as integer coordinates, and the lower 16 bits are the decimal part which this command ignores.

1.239 mgrabparticlesq

```
MGrabParticlesQ CoordinateList.l,NumPoints.l,Buffer.l ; Grab lots of points from X ↩
    .q,Y.q table, to buffer mem
```

Does the same as MGrabParticles except that the list data are pairs of X.q,Y.q. In this respect, a .q quick are longwords each, causing two longwords to be needed for each pixel. The top 16 bits of each component will be used as integer coordinates, and the lower 16 bits are the decimal part which this command will ignore.

1.240 mdrawparticlesq

```
MDrawParticlesQ CoordinateList.l,NumPoints.l,Buffer.l ; Draw previously grabbed ↩
    points, using X.q,Y.q's
```

Does the same as MDrawParticlesQ except using X.q,Y.q list data. Each list item is a longword quick value in which the upper word is taken as the integer coordinate and the lower word, the decimal, is ignored. Two longwords are required for each pixel in the list.

1.241 mgrabparticlesandplotq

```
MGrabParticlesAndPlotQ CoordinateList.l,NumPoints.l,Buffer.l[,Colour.b];Grabs ↩
    points X.q,Y.q to buffer & plots
```

Does the same as MGrabParticlesAndPlot, except using X.q,Y.q list data. Each list item is a longword quick value in which the upper word is taken as the integer coordinate and the lower word, the decimal, is ignored. Two longwords are required for each pixel in the list.

1.242 maddtoparticlesq

```
MAddToParticlesQ CoordinateList.l,NumPoints.l,IncA.l[,IncB.l] ; Add X.q,Y.q to X.q ↩
    ,Y.q items in particle list
```

Does the same as MAddToParticles except that the coordinate list comprises X.q,Y.q pairs of items, and the increment list also contains X.q,Y.q data. If a second increment list is specified, its X.q,Y.q contents will also be added to the data.

1.243 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 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.

1.244 mwrapparticlesa

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

Does the same as MWrapParticles except with a list of Ptr.l data. Because it is not possible to efficiently check the position in terms of X and Y coordinates, the address will only be vertically wrapped. If the pixel goes off the bottom of the bitmap or clip window it should reappear at the top, and vice versa, but going off the left or right edges will not cause a wrap, although some kind of a wrap may occur due to the normal arrangement of the bitmap data.

1.245 mwrapparticlesq

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

Does the same as MWrapParticles except with X.q,Y.q data. Each coordinate pair is two longwords, and the upper word of each longword is taken to be the integer coordinate. The decimal part is ignored. Particles will be wrapped around left, right, top and bottom edges.

1.246 mreboundparticles

```
MReboundParticles CoordinateList.l,NumPoints.l,DirectionList.l,DetectSize.w ; ↩
    Bounce particles off edges
```

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 pairs of coordinates and DirectionList.l points to a list of Xmove.w,Ymove.w 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.

1.247 mreboundparticlesq

```
MReboundParticlesQ CoordinateList.l,NumPoints.l,DiectionList.l,DetectSize.w ; ↩
    Bounce particles off edges
```

Does the same as MReboundParticles except that the coordinate table is a list of X.q,Y.q data and the movement table is a list of Xmove.q,Ymove.q movement adders. The .q quick data are pairs of longwords, the upper word of each being taken as the integer coordinate. Note that the whole of the movement variables will be negated, not just their upper words.

1.248 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, Mc2pCPUmode and M040c2pUsage 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.

1.249 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.

1.250 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.

1.251 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.

1.252 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.