Class Canvas (cont.)

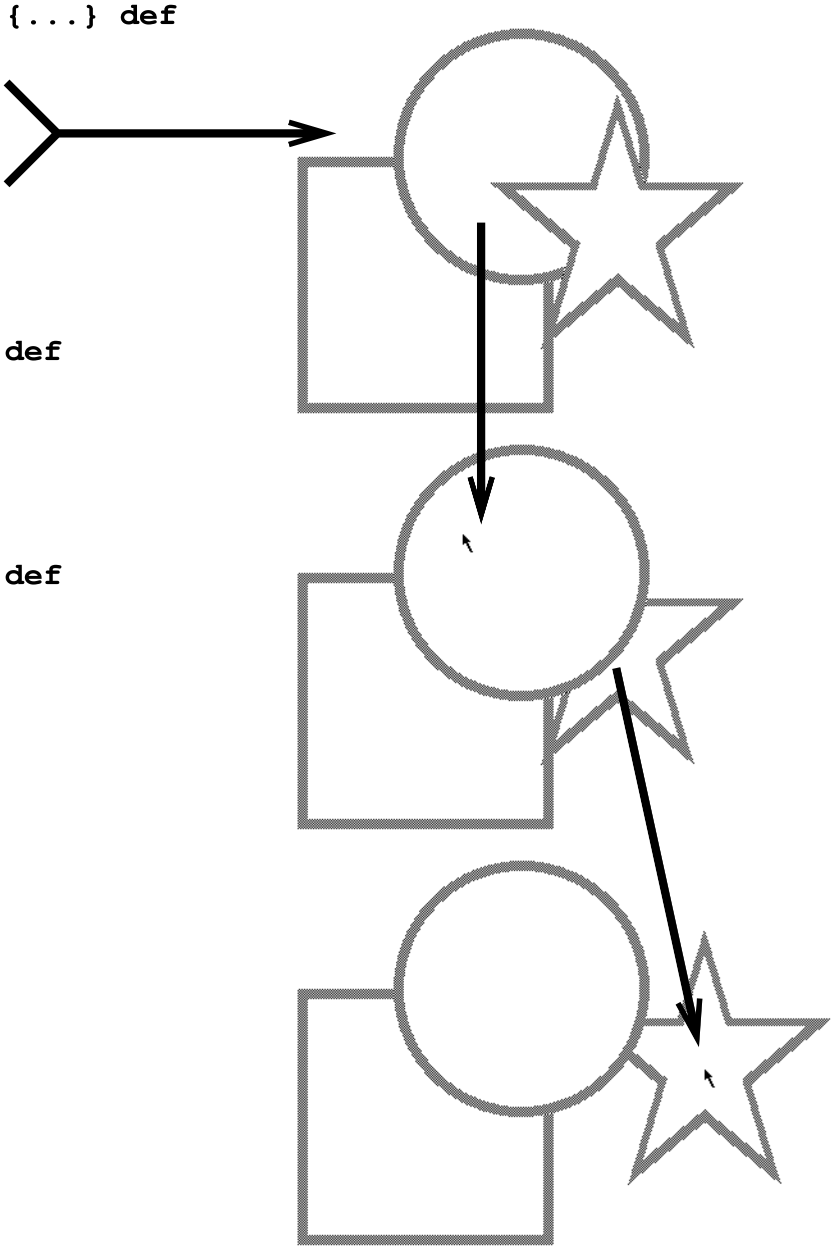
```
/fix { % - => -
gsave
    TheCanvas setcanvas
    damagepath clipcanvas
    /paint self send
    newpath clipcanvas
grestore
} def

/map { % - => -
    EventMgr null eq {/fork self send} if
    TheCanvas /Mapped true put
} def

/fork { % - => -
    /EventMgr [
        PointButton {TheCanvas canvastotop}
        /DownTransition TheCanvas eventmgrinterest

        AdjustButton {TheCanvas false slidecanvas}
        /DownTransition TheCanvas eventmgrinterest

        /Damaged {/fix self send}
        null TheCanvas eventmgrinterest
    ] forkeventmgr def
} def



/demo {
    /can1 framebuffer /new Canvas send def
    50 50 200 200 /reshape can1 send
    /map can1 send

    /can2 framebuffer /new OvalCanvas send def
    125 150 200 200 /reshape can2 send
    /map can2 send

    /can3 framebuffer /new StarCanvas send def
    200 100 200 200 /reshape can3 send
    /map can3 send
} def
```

Class Canvas (cont.)

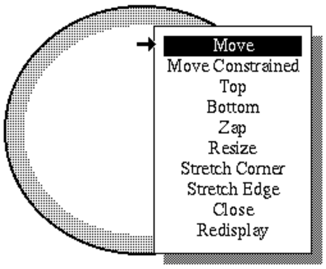
```
/new { % ParentCanvas => instance
    /new super send begin
        /TheCanvas exch newcanvas store
        currentdict
    end
} def

/path {rectpath} def % x y w h => - (set current path)

/reshape { % x y w h => -
gsave
    TheCanvas /Parent get setcanvas
    /Height exch def /Width exch def translate
    0 0 Width Height /path self send
    TheCanvas reshapecanvas
grestore
} def

/reshapefromuser { % - => -
    TheCanvas /Parent get /path true getbbox
    aload pop /reshape self send
} def

/paint { % - => -
gsave
    TheCanvas setcanvas
    EdgeColor fillcanvas FillColor setcolor
    EdgeSize 0 0 Width Height     % delta x y w h
    insetrect /path self send     % -
    fill
grestore
} def
```

A Custom Class: Class Canvas

```
/Canvas Object [/TheCanvas /EventMgr /Height /Width]
classbegin
    /FillColor     1 1 1 rgbcolor def
    /EdgeColor    .5 .5 .5 rgbcolor def
    /EdgeSize      8 def

    /new {...} def
    /path {rectpath} def % x y w h => - (set current path)
    /reshape {...} def
    /reshapefromuser {...} def
    /paint {...} def
    /fix {...} def
    /map {...} def
    /fork {...} def
classend def

/OvalCanvas Canvas []
classbegin
    /path {ovalpath} def
classend def

/StarCanvas Canvas []
classbegin
    /EdgeSize 20 def
    /path {starpath} def
classend def
```

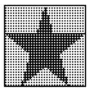


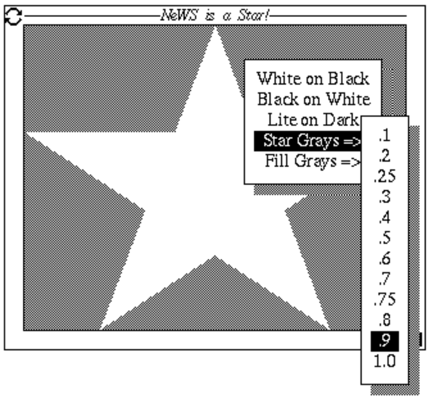
A Custom Window: Oval Window



```
/OvalWindow LiteWindow []
classbegin
    /Border 16 def
    /FrameFillColor .75 .75 .75 rgbcolor def

    /ShapeFrameCanvas { % - => - ([Re]set frame canvas' shape)
        gsave
        ParentCanvas setcanvas
        FrameX FrameY translate
        0 0 FrameWidth FrameHeight ovalpath
        FrameCanvas reshapecanvas
        grestore
    } def
    /PaintFrame        { % - => - (Paint frame canvas)
        FrameFillColor fillcanvas PaintFocus
    } def
    /PaintFocus { % - => - (Paint frame focus)
        gsave
        FrameCanvas setcanvas
        KeyFocus? {0} {FrameFillColor} ifelse setshade
        Border 2 div 0 0 FrameWidth FrameHeight insetrect
        ovalpath stroke
        grestore
    } def
    /ShapeClientCanvas { % - => - ([Re]set client canvas' shape)
        ClientCanvas null ne {
            gsave
            FrameCanvas setcanvas
            Border 0 0 FrameWidth FrameHeight insetrect
            4 2 roll translate 0 0 4 2 roll ovalpath
            ClientCanvas reshapecanvas
            grestore
        } if
    } def
classend def
```

Window & Menu Examples





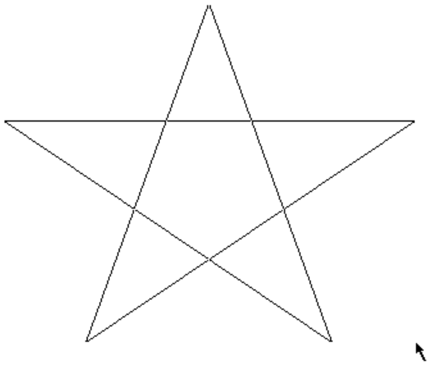
```
#!/usr/NeWS/bin/psh
/StarGray 1 def
/FillGray .5 def
/FillCanvasWithStar { % stargray fillgray => -
    fillcanvas setshade
    clippath pathbbox starpath fill
} def
/SetStarGrays { % stargray fillgray => -
    /FillGray exch store /StarGray exch store
    /paintclient win send
} def

/GetMenuNumber {/currentkey self send cvr} def % - => num
/StarGraysMenu
    [(.1) (.2) (.25) (.3) (.4) (.5) (.6) (.7) (.75) (.8) (.9) (1.0)]
    [{GetMenuNumber FillGray SetStarGrays}]
    /new DefaultMenu send def
/FillGraysMenu
    [(.1) (.2) (.25) (.3) (.4) (.5) (.6) (.7) (.75) (.8) (.9) (1.0)]
    [{StarGray GetMenuNumber SetStarGrays}]
    /new DefaultMenu send def

/win framebuffer /new DefaultWindow send def
{   /FrameLabel (NeWS  is  a  Star!) def
    /PaintIcon {.25 .75 FillCanvasWithStar 0 strokecanvas} def
    /PaintClient {StarGray FillGray FillCanvasWithStar} def
    /ClientMenu [
        (White on Black)   {  1   0 SetStarGrays}
        (Black on White)   {  0   1 SetStarGrays}
        (Lite on Dark)     {.75 .25 SetStarGrays}
        (Star Grays =>)    StarGraysMenu
        (Fill Grays =>)    FillGraysMenu
    ] /new DefaultMenu send def
} win send
/reshapefromuser win send
/map win send
```

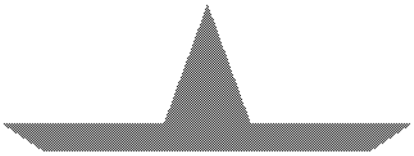
Interact Usage: "slidecanvas"

```
 /slidecanvas { % canvas startup? => - (interactively move canvas)
    gsave
    1 index /Parent get setcanvas
    {gsave dup setcanvas x y movecanvas grestore} exch interact
    pop pop
    grestore
} def



/demo {
    /can1 50 50 200 200 framebuffer rectcanvas def
    can1 setcanvas .5 fillcanvas

    /can2 125 150 200 200 framebuffer ovalcanvas def
    can2 setcanvas .85 fillcanvas

    /can3 200 100 200 200 framebuffer starcanvas def
    can3 setcanvas .75 fillcanvas

    can2 true slidecanvas
} def
```

Interact Usage: "canvasfromuser"

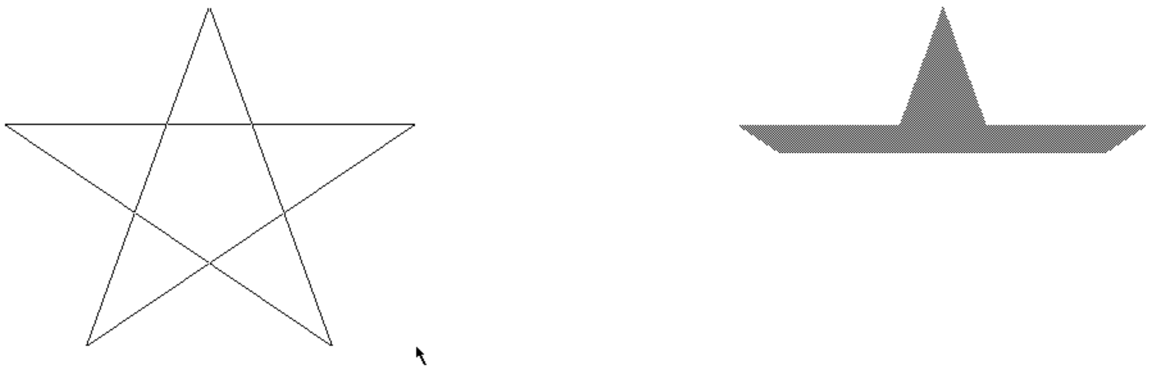
```
/canvasfromuser { % parent pathproc => canvas
    2 copy true getbbox aload pop       % parent proc x y w h
    6 -2 roll pathcanvas                % canvas
} def

/demo {
    /can framebuffer /starpath canvasfromuser def
    can setcanvas .5 fillcanvas
} def
```





Interact Usage: "getbbox"

```
/calcbbox {x0 x min y0 y min x x0 sub abs y y0 sub abs} def
/getbbox { % can pathproc startup? => [x y w h] (relative to can)
    gsave
    3 -1 roll createoverlay setcanvas      % pathproc bool
    {   erasepage calcbbox
        4 index cvx exec stroke            % use the path proc
        Action /UpTransition eq {
            erasepage [calcbbox]
        } if
    } exch interact                        % pathproc array
    exch pop                               % array
    grestore
} def

/demo {
    framebuffer /starpath true getbbox
    /can exch aload pop framebuffer starcanvas def
    can setcanvas .5 fillcanvas
} def
```



User Interaction: "interact" (cont.) MakeInteractInterests

```
/MakeInteractInterests { % - => -
    /StartInterest createevent dup begin
        /Name [ /LeftMouseButton
                /RightMouseButton
                /MiddleMouseButton ] def
        /Action /DownTransition def
        /ClientData {
            /x0 XLocation store /y0 YLocation store
            StopInterest /Name Name put
            TrackInterest expressinterest
            StopInterest expressinterest
            callproc
        } def
    end def

    /TrackInterest createevent dup begin
        /Name /MouseDragged def
        /ClientData {callproc} def
    end def

    /StopInterest createevent dup begin
        /Action /UpTransition def
        /ClientData {callproc exit} def
    end def
} def
```

User Interaction: "interact"

```
/interact { % proc startup? => result
% Repeatedly call proc, with x0 y0 x y defined in a local
% dictionary, whenever MouseDragged & MouseUp.  If startup?
% also call on initial MouseDown.  Returns TOS of tracker.
% Uses "callback" procedures stored in interests.
20 dict begin
    /startup? exch def
    /proc exch cvx def

    currentcursorlocation /y0 exch def /x0 exch def
    /x x0 def /y y0 def
    /callproc {/x XLocation store /y YLocation store proc} def

    MakeInteractInterests

    {   startup?
            {[StartInterest]}
            {proc [TrackInterest StopInterest]} ifelse
        {expressinterest} forall
        {   awaitevent begin
                Interest /ClientData get exec
            end
        } loop
    } fork waitprocess
end
} def

    /p { (x0=% y0=% x=% y=%\n) [x0 y0 x y] printf } def
    /p true interact ==
% x0=463 y0=283 x=463 y=283
% x0=463 y0=283 x=422 y=267
% x0=463 y0=283 x=298 y=193
% x0=463 y0=283 x=238 y=153
% x0=463 y0=283 x=10 y=59
% x0=463 y0=283 x=-64 y=37
% x0=463 y0=283 x=-66 y=35
% x0=463 y0=283 x=-66 y=35
% null
```

Event Example: A Simple Timer

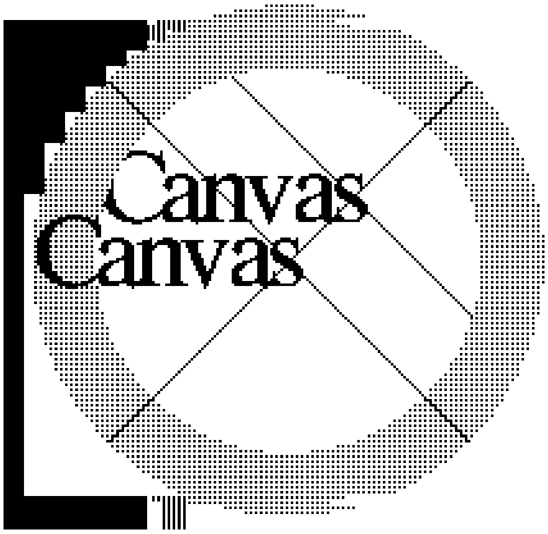
```
/sendtimeevent { % timedelta => -
    createevent begin
        /Name /Timer def
        /TimeStamp exch currenttime add def
        currentdict
    end sendevent
} def

/p {
    /timercount 0 def
    createevent dup /Name /Timer put expressinterest
    {   awaitevent pop
        (Tick\n) print
        /timercount timercount 1 add def
        timercount 10 eq {exit} if
        1 60 div sendtimeevent
    } loop
} fork def

p =
(Starting processes:\n) print
1 60 div sendtimeevent
p waitprocess pop
p =

% process(2716670, runnable)
% Starting processes:
% Tick
% Tick
% Tick
% Tick
% Tick
% Tick
% Tick
% Tick
% Tick
% Tick
% process(2716670, zombie)
```

Process Example: Inherited Environment

```
/a 1 def
/PrintStatus { % - => - (Print current processes status.)
    (State of process:%\n    a=% Dictstack=% stack=)
        [currentprocess a countdictstack] printf
    pstack
} def

PrintStatus

10 dict begin
    /a 10 def
    (Hi!)
    /p {
        PrintStatus
    } fork store
    pop
end

%State of process:process(2222550, runnable)
%    a=1 Dictstack=2 stack=Empty stack

%State of process:process(4041374, runnable)
%    a=10 Dictstack=3 stack=(Hi!) /p
```

Canvas Example: Sub-Canvases

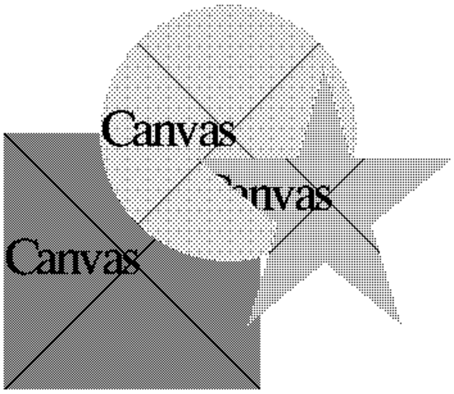
```
/can1 50 50 200 200 framebuffer ovalcanvas def
can1 setcanvas .75 fillcanvas

/can2 25 25 150 150 can1 ovalcanvas def
can2 setcanvas 1 fillcanvas


/Times-Roman findfont 36 scalefont setfont
0 setgray

[can1 can2] {
    setcanvas
    0 90 moveto (Canvas) show
    0 0 moveto 200 200 lineto
    0 200 moveto 200 0 lineto stroke
} forall
```



Canvas Example: Using the "pathcanvas" Utility

```
/can1 50 50 200 200 framebuffer rectcanvas def
can1 setcanvas .5 fillcanvas

/can2 125 150 200 200 framebuffer ovalcanvas def
can2 setcanvas .85 fillcanvas

/can3 200 100 200 200 framebuffer starcanvas def
can3 setcanvas .75 fillcanvas
```

```
/Times-Roman findfont 36 scalefont setfont
0 setgray

[can1 can2 can3] {
    setcanvas
    0 90 moveto (Canvas) show
    0 0 moveto 200 200 lineto
    0 200 moveto 200 0 lineto stroke
} forall
```

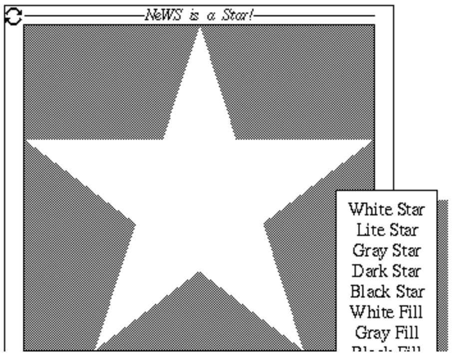


Canvas Example:
The "pathcanvas" Utility

```
/pathcanvas { % x y w h parent path => canvas (make a "path" canvas)
10 dict begin
     gsave
     cvx [/path /parent /h /w /y /x] {exch def} forall
     /can parent newcanvas def

     parent setcanvas
     x y translate 0 0 w h path

     can reshapecanvas
     can /Mapped true put
     can
     grestore
end
} def

/rectcanvas { {rectpath} pathcanvas } def % x y w h parent => canvas
/ovalcanvas { {ovalpath} pathcanvas } def % x y w h parent => canvas
/starcanvas { {starpath} pathcanvas } def % x y w h parent => canvas
```

The NeWS Code Book

- **Code Fragments for Canvases, Processes & Events**
- **A User Interaction Utility**
    - GetBBox, CanvasFromUser & SlideCanvas
- **Class Canvas**
- **LiteWindow & LiteMenu Demos**



NeWS Programming Codebook

10/14/87