EXHIBIT 1

PART 2 OF 2

```
1127
1128    scroll(direction,node)
1129    register char       direction;
1130    register MAPNODE    *node;
1131    {
1132        register char   *reply;
1133        register short  low_row, low_col, pict_ht, pict_wd, *p;
1134        short           map_row, map_col;
1135
1136        if (node && node->picture.pid && node->window.pid && !node->metaphor)
1137            if (reply = Call(DIRECT,node->window.pid,Newmsg(64,"query",NULL),0,0))
1138            {
1139                if (p = (short *) Find_triple(reply,"view",0,NULL,4,NULL))
1140                {
1141                    map_row = *p++;
1142                    map_col = *p;
1143                    Free(reply);
1144                    reply =
1145                        Call(DIRECT,node->picture.pid,Newmsg(32,"query",NULL),0,0);
1146                    p = (short *) Find_triple(reply,"size",0,NULL,4,NULL);
1147                    pict_ht = *p++;
1148                    pict_wd = *p;
1149                    p = (short *) Find_triple(reply,"low ",0,NULL,4,NULL);
1150                    low_row = *p++;
1151                    low_col = *p;
1152                    scroll_pos(node,direction,
1153                        &map_row,&map_col,low_row,low_col,pict_ht,pict_wd);
1154                    Put(DIRECT,node->window.pid,Newmsg(64,"map",
1155                        "to=#C; at=#2s",&node->picture,map_row,map_col));
1156                }
1157                Free(reply);
1158            }
1159    }
```

```
1160
1161    scroll_pos(node,direction,map_row,map_col,low_row,low_col,pict_ht,pict_wd)
1162    register MAPNODE    *node;
1163    register char       direction;
1164    register short      low_row, low_col, pict_ht, pict_wd, *map_row, *map_col;
1165    {
1166        switch (direction)
1167        {
1168            case 'u':   if (*map_row - low_row >= VCHAR_HT)
1169                            *map_row -= VCHAR_HT;
1170                        break;
1171            case 'd':   if (pict_ht - (*map_row-low_row) - node->height >= VCHAR_HT)
1172                            *map_row += VCHAR_HT;
1173                        break;
1174            case 'l':   if (*map_col - low_col >= VCHAR_WD)
1175                            *map_col -= VCHAR_WD;
1176                        break;
1177            case 'r':   if (pict_wd - (*map_col-low_col) - node->width >= VCHAR_WD)
1178                            *map_col += VCHAR_WD;
1179                        break;
1180            case 'U':   if (*map_row - low_row >= node->height)
1181                            *map_row -= node->height;
1182                        else
1183                            *map_row = low_row;
1184                        break;
1185            case 'D':   if (pict_ht - (*map_row - low_row) >= 2 * node->height)
1186                            *map_row += node->height;
1187                        else
1188                            *map_row = pict_ht - low_row - node->height;
1189                        break;
1190            case 'L':   if (*map_col - low_col >= node->width)
1191                            *map_col -= node->width;
1192                        else
1193                            *map_col = low_col;
1194                        break;
1195            case 'R':   if (pict_wd - (*map_col - low_col) >= 2 * node->width)
1196                            *map_col += node->width;
1197                        else
1198                            *map_col = pict_wd - low_col - node->width;
1199        }
1200    }
```

```
1201
1202    notify_process(node,row,col,act,area,hdr,indic,active)
1203    register MAPNODE    *node;
1204    register P_E_HDR    *hdr;
1205    register char       act, area;
1206    char                indic;
1207    short               row, col;
1208    MAPNODE             *active;
1209    {
1210        register char    *p, *m;
1211        register int     len = 0;
1212
1213        if (hdr)
1214            len = *(short *) hdr;
1215        m = Newmsg(len+200,"click",
1216            "from=#C; map=#C; name=#S; actn=#b; what=#b; pos=#2s",
1217            &node->window,&node->picture,node->name,act,area,row,col);
1218        if (hdr)
1219        {
1220            p = Append_triple(m,"data",len+6,hdr);
1221            ((P_E_HDR*)p)->attr.selected = NO;
1222            p += *(short *) p;
1223            Long_align(p);
1224            *(short *) p = NULL;
1225        }
1226        if (indic)
1227            Append_triple(m,"char",1,&indic);
1228        if (active)
1229            Append_triple(m,"acty",4,&active->owner);
1230        Put(DIRECT,node->owner.pid,m);
1231    }
```

```
1232
1233    Metaphor(screen,map,buf,size,output,dialogue)
1234    register SCREEN     *screen;
1235    register LIST       *map;
1236    register long       buf, size, output;
1237    CONNECTOR           *dialogue;
1238    {
1239        register short      *p;
1240        register MAPNODE    *node;
1241
1242        screen->meta_row = screen->meta_col = 0;
1243        screen->meta_ht = screen->height;
1244        screen->meta_wd = screen->width;
1245        if (node = create_window(screen,map,output,"Metaphor",buf,size))
1246        {
1247            map->metaphor = node;
1248            node->owner = *dialogue;
1249            p = (short *) Find_triple(buf,"area",size,none,8,NULL);
1250            screen->meta_row = *p++;
1251            screen->meta_col = *p++;
1252            screen->meta_ht = *p++;
1253            screen->meta_wd = *p;
1254            node->metaphor = node->never = node->keep_open = YES;
1255            node->fixed = node->nonmod = YES;
1256            Reply(buf,Newmsg(32,"connect","conn=#C",&node->window));
1257        }
1258        else
1259            reply_status(buf,"-Metaphor","can\'t create \'window\'",0);
1260    }
```

```
1261
1262    MAPNODE *create_terminal(screen,map,output,buf,size,sender)
1263    SCREEN              *screen;
1264    register LIST       *map;
1265    CONNECTOR           *output;
1266    register long       buf, size, sender;
1267    {
1268        static char         def_type[] = "//processes/terminal";
1269        register MAPNODE    *node;
1270        register char       *p;
1271        CONNECTOR           terminal;
1272
1273        if (Find_triple(buf,"name",size,NULL,1,NULL))
1274        {
1275            if (terminal.pid = NewProc("Terminal",
1276                Find_triple(buf,"emul",size,def_type,1,NULL),YES,-1))
1277            {
1278                p = Alloc(size,YES);
1279                memcpy(p,buf,size);
1280                memcpy(p,sender,sizeof(CONNECTOR));
1281                memcpy(p+sizeof(CONNECTOR),&terminal,sizeof(CONNECTOR));
1282                p = Call(DIRECT,terminal.pid,p,0,0);
1283                if (!strcmp(p,"create")
1284                    && (node = create_window(screen,map,output,"Window",p,size)))
1285                {
1286                    node->terminal = node->owner = terminal;
1287                    Free(p);
1288                    return(node);
1289                }
1290            }
1291            reply_status(buf,"-create","can\'t create \'terminal\'",0);
1292        }
1293        else
1294            reply_status(buf,"-create","(terminal) no name given",0);
1295        return(NULL);
1296    }
```

```
1297
1298    MAPNODE *create_window(screen,map,output,proc,buf,size)
1299    SCREEN                  *screen;
1300    LIST                    *map;
1301    CONNECTOR               *output;
1302    char                    *proc;
1303    register long           buf, size;
1304    {
1305        static char             def_outl[4] = {GREEN,3,BLACK,'S'};
1306        register char           *window_name, *title, *p;
1307        register short          pict_row = 0, pict_col = 0;
1308        register MAPNODE        *node;
1309        char                    out_clr, out_fill, pane_clr;
1310        MAPNODE                 *new_node();
1311
1312        if ((window_name = Find_triple(buf,"name",size,NULL,1,NULL))
1313            && (node = new_node(map,window_name))
1314            && (node->window.pid = NewProc(proc,"//processes/window",YES,-1)))
1315        {
1316            map_after(node,NULL,map);
1317            title = Find_triple(buf,"titl",size,window_name,1,NULL);
1318            init_node(node,buf,size);
1319            strcpy(node->device,Find_triple(buf,"from",size,none,2,NULL));
1320            strncpy(node->term,
1321                Find_triple(buf,"mod ",size,none,1,NULL),sizeof(node->term)-1);
1322            strncpy(node->special,
1323                Find_triple(buf,"spec",size,none,1,NULL),sizeof(node->special)-1);
1324            p = Find_triple(buf,"outl",size,def_outl,4,NULL);
1325            out_clr = *p++;
1326            node->outline = *p++;
1327            if (!(out_fill = *p++))
1328                out_fill = BLACK;
1329            if (!(node->style = *p))
1330                node->style = 'S';
1331            node->pane = 0;
1332            pane_clr = out_clr;
1333            if (p = Find_triple(buf,"pane",size,NULL,2,NULL))
1334            {
1335                pane_clr = *p++;
1336                node->pane = *p;
1337            }
1338            else if (node->Hscroll || node->Vscroll)
1339                node->pane = 1;
1340            if (p = Find_triple(buf,"map ",size,NULL,8,NULL))
1341            {
1342                node->picture = *(CONNECTOR *) p;
1343                if (*(long*)(p-4) > sizeof(CONNECTOR))
1344                {
```

```
1345                    pict_row = *(short *) (p + sizeof(CONNECTOR));
1346                    pict_col = *(short *) (p + sizeof(CONNECTOR) + sizeof(short));
1347                }
1348            }
1349            if (init_window(screen,node,output,title,pict_row,pict_col,
1350                        out_clr,out_fill,0,pane_clr))
1351            {
1352                activate(node);
1353                clip_window(map->last);
1354                return(node);
1355            }
1356        }
1357        reply_status(buf,"-create","(window)",0);
1358        return(NULL);
1359    }
1360
1361    init_node(node,buf,size)
1362    register MAPNODE    *node;
1363    register long       buf, size;
1364    {
1365        static short    def_pos[2] = {0,0}, def_size[2] = {5,10};
1366        register char    *p;
1367
1368        p = Find_triple(buf,"pos ",size,def_pos,4,NULL);
1369        node->row = *((short *) p)++;
1370        node->col = *(short *) p;
1371        p = Find_triple(buf,"size",size,def_size,4,NULL);
1372        node->out_ht = node->height = *((short *) p)++;
1373        node->out_wd = node->width = *(short *) p;
1374        node->title = check_bar(buf,"tbar",VCHAR_HT);
1375        node->menu = check_bar(buf,"mbar",VCHAR_HT);
1376        node->Vscroll = check_bar(buf,"vbar",YES);
1377        node->Hscroll = check_bar(buf,"hbar",YES);
1378        node->general_use = check_bar(buf,"gbar",YES);
1379        node->corner = check_bar(buf,"corn",YES);
1380        node->resize_box = check_bar(buf,"rsiz",YES);
1381        if (node->palette = check_bar(buf,"pbar",5*VCHAR_WD))
1382            node->palette += 2 *VCHAR_WD;
1383        window_options(node,buf,size);
1384    }
```

```
1385
1386    check_bar(ptr,keyw,deflt)
1387    register char   *ptr, *keyw;
1388    register short  deflt;
1389    {
1390        register short  *p;
1391
1392        if (!(p = (short *) Find_triple(ptr,keyw,0,NO,0,NULL)))
1393            return(NO);
1394        else if (p == (short *) 1)
1395            return(deflt);
1396        else
1397            return(*p);
1398    }
```

```
1399
1400    window_options(node,buf,size)
1401    register MAPNODE    *node;
1402    register long       buf, size;
1403    {
1404        register char   *options, opt;
1405
1406        options = Find_triple(buf,"when",size,none,1,NULL);
1407        while (opt = *options++)
1408            switch (opt)
1409            {
1410                case 'S':   node->on_element = opt;  break;
1411                case 'X':   node->on_cancel = opt;  break;
1412                case 's':   node->on_select = opt;  break;
1413                case 'O':   node->on_open = opt;  break;
1414                case 'M':   node->on_modify = opt;  break;
1415                case 'C':   node->on_close = opt;  break;
1416                case 'Q':   node->on_quit = opt;  break;
1417                case 'W':   node->on_window_edge = opt;  break;
1418                case 'P':   node->on_picture_edge = opt;  break;
1419                case 'A':   node->on_anychar = opt;  break;
1420                case 'D':   node->on_delete = opt;  break;
1421                case 'B':   node->on_box = opt;  break;
1422                case 'L':   node->on_location = opt;  break;
1423                case 'N':   node->on_insert = opt;  break;
1424            }
1425        options = Find_triple(buf,"opt ",size,none,1,NULL);
1426        while (opt = *options++)
1427            switch (opt)
1428            {
1429                case 'H':   node->auto_highlight = opt;  break;
1430                case 'E':   node->editable = opt;  break;
1431                case 'S':   node->multi_select = opt;  break;
1432                case 'X':   node->never = opt;  break;
1433                case 'B':   node->remap = opt;  break;
1434                case 'N':   node->nonmod = opt;  break;
1435                case 'F':   node->fixed = opt;  break;
1436                case 'O':   node->keep_open = opt;  break;
1437                case 'M':   node->move_mark = opt;  break;
1438                case '+':   node->tight = opt;  break;
1439                case '-':   node->picture.pid = NULL;
1440            }
1441    }
```

```
1442
1443    init_window(screen,node,output,title,row,col,out_clr,out_fill,out_pat,pane_clr)
1444    register SCREEN     *screen;
1445    register MAPNODE    *node;
1446    CONNECTOR           *output;
1447    register short      row, col;
1448    register char       *title;
1449    register char       out_clr, out_fill, out_pat, pane_clr;
1450    {
1451        register char   *msg;
1452        int             result = NO;
1453
1454        if (node->style == 's' && (screen->colors < 7 || !screen->bit_map))
1455            node->style = 'S';
1456        if (node->outline)
1457            out_line(node);
1458        if (node->palette)
1459            node->left = node->palette;
1460        if (node->resize_box || node->Vscroll)
1461            node->right += VCHAR_WD;
1462        if (node->corner && !node->palette)
1463            node->left += VCHAR_WD;
1464        if (node->menu || node->general_use)
1465            node->bottom = VCHAR_HT * 2;
1466        else if (node->Hscroll)
1467            node->bottom = VCHAR_HT;
1468        align_window(screen,node);
1469        msg = Newmsg(3000,"init",
1470            "pos=#2s; size=#2s; outl=#5b; pane=#2b; marg=#4s; scrn=#4s; outp=#C; \
1471            self=#C; map=#C#2s; name=#S; rply",
1472            node->row,node->col,node->height,node->width,
1473            out_clr,node->outline,out_fill,out_pat,node->style,pane_clr,node->pane,
1474            node->top,node->bottom,node->left,node->right,0,0,screen->height,
1475            screen->width,output,&node->window,&node->picture,row,col,node->name);
1476        init_frame(msg,node,title,out_clr);
1477        msg = Call(DIRECT,node->window.pid,msg,0,0);
1478        result = strcmp(msg,"failed");
1479        Free(msg);
1480        return(result);
1481    }
```

```
1482
1483    out_line(node)
1484    register MAPNODE    *node;
1485    {
1486        node->outer = node->outline + node->pane + (node->outline && node->pane) *
1487                (node->height/100 + node->width/100 + 2);
1488        if (node->tight)
1489        {
1490            node->top = node->bottom = node->outer;
1491            node->left = node->right = node->outer+ node->width/200;
1492        }
1493        else
1494        {
1495            node->top = VCHAR_HT;
1496            node->bottom = node->outer;
1497            node->left = node->right = VCHAR_WD;
1498        }
1499        if (node->style == 's')
1500        {
1501            node->bottom += 5;
1502            node->right += 5;
1503        }
1504    }
```

```
1505
1506    init_frame(msg,node,title,out_clr)
1507    register MAPNODE    *node;
1508    register char       *msg, *title, out_clr;
1509    {
1510        char                *n, *frame_bar();
1511        register char       scroll_clr = (4*16)+YELLOW, title_clr = WHITE;
1512        register P_E_HDR    *hdr;
1513        static short        up_arrow[] = {7,0,0,6,7,12,7,9,10,9,10,3,7,3,7,0};
1514        static short        down_arrow[] = {3,0,3,3,0,3,0,9,3,9,3,12,10,6,3,0};
1515        static short        left_arrow[] = {6,0,0,7,3,7,3,10,9,10,9,7,12,7,6,0};
1516        static short        right_arrow[] = {3,0,3,3,0,3,6,10,12,3,9,3,9,0,3,0};
1517        static long         resize_symbol[] = {0x00007f80,0x7f807f80,0x7f807f80,
1518                            0x7f807fc4,0x00ec007c,0x003c007c,0x00fc0000,0x00000000};
1519
1520        if (node->title)
1521        {
1522            n = frame_bar(msg,"top",400,'T',0,0,node->title,1000,0,out_clr,0,NO);
1523            draw_filled_rect(&n,0,0,node->title,1000,NULL,0,0,out_clr,0,0,0,0,"a");
1524            draw_rect(&n,5,10,10,10,"CLOSE!",title_clr,'S',1,"S");
1525            draw_text(&n,0,3*VCHAR_WD,title,"NAME",title_clr,0,NULL,NULL);
1526            if (!node->nonmod)
1527            {
1528                hdr = (P_E_HDR *) start_macro(&n,0,1000,
1529                    VCHAR_HT-2,2*VCHAR_WD,'N',"FILL!",0,0,"Sa");
1530                draw_rect(&n,3,0,VCHAR_HT-7,2*VCHAR_WD-4,NULL,title_clr,'S',1,NULL);
1531                draw_filled_rect(&n,6,3,VCHAR_HT-14,2*VCHAR_WD-10,
1532                    NULL,0,0,title_clr,0,0,0,0,NULL);
1533                end_macro(&n,hdr);
1534            }
1535            draw_end(&n);
1536        }
1537        if (node->Vscroll)
1538        {
1539            n = frame_bar(msg,"rght",400,'V',node->pane-1,node->pane-1,790,
1540                    node->right-node->pane-node->outline+2,out_clr,BLACK,1,NO);
1541            draw_rect(&n,node->pane,node->pane,VCHAR_HT-4,
1542                node->right-(node->pane)-(node->outline),
1543                "SCROLL!",scroll_clr,'S',1,"Sb");
1544            draw_poly(&n,875,node->pane+1,
1545                8,up_arrow,"UP!",scroll_clr,0,0,'S',0,1,"Sa");
1546            draw_poly(&n,980,node->pane+1,
1547                8,down_arrow,"DOWN!",scroll_clr,0,0,'S',0,1,"Sa");
1548            draw_end(&n);
1549        }
```

```
1550
1551        if (node->Hscroll)
1552        {
1553            n = frame_bar(msg,"bot ",400,'H',node->pane-1,0,
1554                    node->bottom-(node->pane)-(node->outline)+2,
1555                    910,out_clr,BLACK,1,NO);
1556            draw_rect(&n,node->pane,node->pane,
1557                node->bottom-node->pane-node->outline,
1558                2*VCHAR_WD-2,"SCROLL!",scroll_clr,'S',1,"Sb");
1559            draw_poly(&n,node->pane,955,
1560                8,left_arrow,"LEFT!",scroll_clr,0,0,'S',0,1,"Sa");
1561            draw_poly(&n,node->pane,990,
1562                8,right_arrow,"RIGHT!",scroll_clr,0,0,'S',0,1,"Sa");
1563            draw_end(&n);
1564        }
1565        if (node->menu)
1566            frame_bar(msg,"bot ",200,'M',node->pane-1,0,
1567                node->bottom-(node->pane)-(node->outline)+2,
1568                1000,out_clr,BLACK,1,YES);
1569        if (node->general_use)
1570            frame_bar(msg,"bot ",200,'G',node->pane-1,0,
1571                node->bottom-(node->pane)-(node->outline)+2,
1572                1000,out_clr,BLACK,1,YES);
1573        if (node->palette)
1574            frame_bar(msg,"left",200,'P',0,node->pane,10000,
1575                node->left-(node->pane)-(node->outline)-1,out_clr,BLACK,1,YES);
1576        if (node->resize_box)
1577        {
1578            n = frame_bar(msg,"rbox",200,NULL,0,0,0,0,scroll_clr,BLACK,1,NO);
1579            draw_symbol(&n,0,0,16,16,resize_symbol,"RESIZE!",scroll_clr,0,"S");
1580            draw_end(&n);
1581        }
1582        if (node->corner)
1583            frame_bar(msg,"lbox",200,NULL,0,0,0,0,out_clr,BLACK,1,YES);
1584    }
```

```
1585
1586    char *frame_bar(msg,keyw,size,type,row,col,height,width,color,fill,thick,end)
1587    register char     *msg, *keyw;
1588    char              type, color, fill, end;
1589    register short    row, col, height, width, size, thick;
1590    {
1591        char    *n;
1592
1593        n = Append_triple(msg,keyw,size,NULL);
1594        *n++ = type;
1595        draw_filled_rect(&n,row,col,height,width,
1596            NULL,color,0,fill,0,'S',0,thick,"a");
1597        if (end)
1598            draw_end(&n);
1599        return(n);
1600    }
1601
1602    Set_user(name,buf,size)
1603    register NAME     *name;
1604    register long     buf, size;
1605    {
1606        register char   *p;
1607
1608        if (p = Find_triple(buf,"name",size,NULL,2,NULL))
1609        {
1610            strcpy(name->user,p);
1611            Note("signed on",p);
1612            Put(ALL,"HI",Newmsg(128,"U","name=#S",p));
1613        }
1614    }
```

```
1615
1616    Change(screen,map,msg)
1617    SCREEN     *screen;
1618    LIST       *map;
1619    MESSAGE    *msg;
1620    {
1621        register CONNECTOR  *window, *owner = NULL;
1622        register short      *p;
1623        register MAPNODE    *node;
1624
1625        if (window = (CONNECTOR*)Find_triple(msg->buf,"conn",msg->size,NULL,8,NULL))
1626        {
1627            for (node = map->first; node && node->window.pid != window->pid
1628                && node->terminal.pid != window->pid; node = node->nxt) ;
1629            if (node)
1630            {
1631                if (p = (short*) Find_triple(msg->buf,"size",msg->size,none,4,NULL))
1632                    resize(screen,node,*p,*(p+1));
1633                if (Find_triple(msg->buf,"actv",msg->size,NULL,0,NULL)
1634                    && !node->never)
1635                        map->active = node;
1636                if (owner =
1637                    (CONNECTOR*) Find_triple(msg->buf,"ownr",msg->size,NULL,0,NULL))
1638                        if ((long)owner == 1)
1639                            owner = &msg->sender;
1640                if (owner)
1641                {
1642                    node->owner = *owner;
1643                    if (node->terminal.pid)
1644                    {
1645                        Forward(DIRECT,node->terminal.pid,msg->buf);
1646                        msg->buf = NULL;
1647                    }
1648                }
1649                clip_window(map->last);
1650            }
1651        }
1652    }
```

```
1653
1654    highlight(node,map)
1655    register MAPNODE    *node;
1656    register LIST       *map;
1657    {
1658        if (node && node != map->last_active)
1659        {
1660            if (!node->metaphor)
1661            {
1662                Put(LOCAL,"Window",
1663                    Newmsg(64,"highlight","bar=#b; tag=#S",'T',"CLOSE!"));
1664                if (node->window.pid && node->title)
1665                    Put(DIRECT,node->window.pid,
1666                        Newmsg(128,"highlight","off; bar=#b; tag=#S",'T',"CLOSE!"));
1667            }
1668            if (node->window.pid)
1669                Put(DIRECT,node->window.pid,Newmsg(32,"keys?",NULL));
1670            map->last_active = node;
1671        }
1672    }
1673
1674    move_mark(row,col,picture)
1675    register short      row, col;
1676    register CONNECTOR  *picture;
1677    {
1678        Put(DIRECT,picture->pid,Newmsg(32,"mark","at=#2s",row,col));
1679    }
1680
```

```
1681
1682    clip_window(node)
1683    register MAPNODE    *node;
1684    {
1685        register MAPNODE    *temp;
1686        register short      prio = 127, count, *count_addr, *n;
1687        char                *m;
1688
1689        for ( ; node; node = node->pre)
1690        {
1691            m = Newmsg(1000,"cut","inHI=#s#s#A",prio--,0,950,NULL);
1692            count_addr = (short *) (Find_triple(m,"inHI",0,NULL,0,NULL) + 2);
1693            n = count_addr + 1;
1694            count = 0;
1695            for (temp = node->pre; temp; temp = temp->pre)
1696            {
1697                *n++ = temp->row;
1698                *n++ = temp->col;
1699                *n++ = temp->out_ht;
1700                *n++ = temp->out_wd;
1701                count++;
1702            }
1703            *count_addr = count;
1704            Put(DIRECT,node->window.pid,m);
1705        }
1706    }
1707
1708    MAPNODE *find_window(map,window,row,col)
1709    register LIST   *map;
1710    register WINDOW *window;
1711    register short  row, col;
1712    {
1713        register MAPNODE    *node;
1714
1715        for (node = map->first; node; node = node->nxt)
1716        {
1717            query_window(window,node->window,row,col);
1718            if (window->area != 'N')
1719                break;
1720        }
1721        window->previous = window->node;
1722        return(window->node = node);
1723    }
```

```
1724
1725    query_window(window,conn,row,col)
1726    register WINDOW *window;
1727    CONNECTOR       conn;
1728    register short  row, col;
1729    {
1730        register char   *p, *reply;
1731
1732        if (window->hdr)
1733            Free(window->hdr);
1734        window->hdr = NULL;
1735        window->elem_row = window->elem_col = -1;
1736        reply = Call(DIRECT,conn.pid,Newmsg(64,"w","inHI=#2s",row,col),0,0);
1737        p = Find_triple(reply,"inHI",0,none,1,NULL);
1738        p += 2 *sizeof(short);
1739        window->area = *p++;
1740        window->bar = *p++;
1741        window->row = *((short *) p)++;
1742        window->col = *((short *) p)++;
1743        Long_align(p);
1744        if (*(short*)p)
1745        {
1746            window->hdr = (P_E_HDR *) Alloc(*(short*)p,YES);
1747            memcpy(window->hdr,p,*(short*)p);
1748        }
1749        Free(reply);
1750    }
1751
1752    MAPNODE *new_node(map,name)
1753    register LIST   *map;
1754    register char   *name;
1755    {
1756        register MAPNODE    *node = NULL;
1757        register short      i;
1758
1759        for (i = POOL_SIZE, node = map->pool; node->pool && i; ++node, --i) ;
1760        if (!i)
1761            node = (MAPNODE *) Alloc(sizeof(MAPNODE),YES);
1762        memset(node,0,sizeof(MAPNODE));
1763        node->pool = i;
1764        strcpy(node->name,name);
1765        return(node);
1766    }
```

```
1767
1768    free_node(node)
1769    register MAPNODE    *node;
1770    {
1771        if (node->pool)
1772            node->pool = NULL;
1773        else
1774            Free(node);
1775    }
1776
1777    map_after(node,pred,map)
1778    register MAPNODE    *node, *pred;
1779    register LIST       *map;
1780    {
1781        if (pred)
1782        {
1783            node->nxt = pred->nxt;
1784            node->pre = pred;
1785            if (pred->nxt)
1786                (pred->nxt)->pre = node;
1787            pred->nxt = node;
1788        }
1789        else
1790        {
1791            if (node->nxt = map->first)
1792                (map->first)->pre = node;
1793            node->pre = NULL;
1794        }
1795        if (!node->pre)
1796            map->first = node;
1797        if (!node->nxt)
1798            map->last = node;
1799        ++map->count;
1800    }
1801
1802    unmap(node,map)
1803    register MAPNODE    *node;
1804    register LIST       *map;
1805    {
1806        if (node->pre)
1807            (node->pre)->nxt = node->nxt;
1808        else
1809            map->first = node->nxt;
1810        if (node->nxt)
1811            (node->nxt)->pre = node->pre;
1812        else
1813            map->last = node->pre;
1814        --map->count;
1815    }
```

```
1816
1817    remap(window,node,new_picture,map,sel)
1818    register CONNECTOR    *window, *new_picture;
1819    register MAPNODE      *node;
1820    register SELECTION    *sel;
1821    LIST                  *map;
1822    {
1823        if (window)
1824            for (node = map->first;
1825                 node && window->pid != node->window.pid; node = node->nxt) ;
1826        if (node)
1827        {
1828            end_edit(node,'X',0,0,NULL);
1829            if (new_picture && new_picture->pid != node->picture.pid)
1830                if (node == sel->map)
1831                {
1832                    sel->map = NULL;
1833                    sel->pending = NO;
1834                }
1835            node->picture = *new_picture;
1836        }
1837    }
```

```
1838
1839    align_window(screen,node)
1840    register SCREEN     *screen;
1841    register MAPNODE    *node;
1842    {
1843        register short  temp;
1844
1845        if (screen->char_align)
1846        {
1847            if (node->tight)
1848            {
1849                temp = ((node->row % VCHAR_HT) || node->outer ? VCHAR_HT : 0);
1850                node->row = (node->row / VCHAR_HT) * VCHAR_HT - node->outer + temp;
1851                temp = ((node->col % VCHAR_WD) || node->outer ? VCHAR_WD : 0);
1852                node->col = (node->col / VCHAR_WD) * VCHAR_WD -
1853                            (node->outer + node->width/200) + temp;
1854            }
1855            else
1856            {
1857                node->row = ((node->row + VCHAR_HT-1) / VCHAR_HT) * VCHAR_HT;
1858                node->col = ((node->col + VCHAR_WD-1) / VCHAR_WD) * VCHAR_WD;
1859            }
1860            if (node->row < screen->meta_row)
1861                node->row += (screen->meta_row + VCHAR_HT-1) / VCHAR_HT * VCHAR_HT;
1862            if (node->col < screen->meta_col)
1863                node->col += (screen->meta_col + VCHAR_WD-1) / VCHAR_WD * VCHAR_WD;
1864            if (node->out_ht > screen->meta_ht)
1865                node->height = screen->meta_ht - (node->top + node->bottom);
1866            if (node->out_wd > screen->meta_wd)
1867                node->width = screen->meta_wd - (node->left + node->right);
1868            if (!node->tight)
1869            {
1870                temp = (node->height % VCHAR_HT ? VCHAR_HT : 0);
1871                node->height = (node->height/VCHAR_HT) * VCHAR_HT + temp;
1872                temp = (node->width % VCHAR_WD ? VCHAR_WD : 0);
1873                node->width = (node->width/VCHAR_WD) * VCHAR_WD + temp;
1874            }
1875        }
1876        node->out_ht = node->height + node->top + node->bottom;
1877        node->out_wd = node->width + node->left + node->right;
1878    }
```

```
1879
1880    Status(msg,size)
1881    register char   *msg;
1882    register long   size;
1883    {
1884        register char   *m;
1885
1886        *(m = Alloc(size,YES)) = NULL;
1887        strcat(m,Find triple(msg,"orig",size,none,1,NULL));
1888        strcat(m," ");
1889        strcat(m,Find triple(msg,"stat",size,none,1,NULL));
1890        strcat(m," in");
1891        strcat(m,Find triple(msg,"req ",size,none,1,NULL));
1892        Note(m,"ERROR");
1893        Free(m);
1894    }
1895
1896    reply status(req,mid,stat,code)
1897    register char    *req, *mid, *stat;
1898    register long    *code;
1899    {
1900        register char   *type, *msg;
1901
1902        type = "failed";
1903        if (!mid)
1904            type = "status";
1905        else if (*mid == '-')
1906            mid++;
1907        else if (*mid == '+')
1908        {
1909            type = "done";
1910            mid++;
1911        }
1912        msg = Newmsg(strlen(stat)+100,type,
1913            "orig=#S; stat=#S; code=#l","console",stat,code);
1914        if (mid)
1915        {
1916            Append triple(msg,"req ",strlen(mid)+1,mid);
1917            Reply(req,msg);
1918        }
1919        else
1920            Put(DIRECT,(long)req,msg);
1921    }
1922
1923    info(dialogue,string,window)
1924    CONNECTOR       dialogue, window;
1925    register char   *string;
1926    {
1927        Put(DIRECT,dialogue,pid,Newmsg(strlen(string)+100,"info",
1928            "text=#S; near=#C; wait=#s",string,&window,5));
1929    }
```

```
 9        Module              : %M% %I%
10        Date submitted      : %E% %U%
11        Author              : Frank Kolnick
12        Origin              : cX
13        Description         : Picture Manager
14
15   ************************************************************************/
16
17   #ifndef lint
18   static char SrcId[] = "%Z% %M%:%I%";
19   #endif
20   /* Picture manager:  global data */
21
22   #include    <cX.h>                                  /* cX definitions */
23   #include    <HI.h>                                  /* picture, etc. definitions */
24   #include    <memory.h>
25   #include    <string.h>
26   static long none = 0;
27
28   typedef struct element_node                         /* links picture elements: */
29   {
30       struct element_node *nxt;                       /*      ->next node */
31       struct element_node *pre;                       /*      ->preceding node */
32       unsigned char       changed;                    /*      'element has changed' */
33       unsigned char       marked;                     /*      'element is marked' */
34       unsigned char       deleted;                    /*      'no longer in use' */
35       unsigned char       pool;                       /*      'local buffer pool' */
36       short               length;                     /*      (start of element) */
37       /***** NOTE: 'length' must start on a long-word boundary *****/
38   } ELEMENT;
39
40   typedef struct current_state                        /* current data: */
41   {
42       char                *msg;                       /*      ->current msg. */
43       CONNECTOR           sender;                     /*      conn. to msg. sender */
44       long                size;                       /*      size of msg. */
45       long                appl;                       /*      relevant application */
46       short               appl_row, appl_col;         /*      application 'origin' */
47       CONNECTOR           owner;                      /*      conn. to owning proc. */
48       char                *mark;                      /*      current mark element */
49       char                *old_mark;                  /*      copy of previous mark */
50       char                *erase_mark;                /*      element to erase mark */
51       unsigned char       display_mark;               /*      'display mark' */
52       unsigned char       private;                    /*      'private picture' */
53       unsigned char       check;                      /*      'check size' */
54       unsigned char       debug;                      /*      'print diagnostics' */
55       char                highlight;                  /*      type of highlighting */
56       char                name[32];                   /*      picture's name */
57       char                file[64];                   /*      picture file's name */
58       long                status_code;                /*      current status ... */
59       char                *status_string;             /*      ... */
60   } CURRENT;
```

```
61
62
63  typedef struct view_node                        /* links viewports: */
64  {
65      struct view_node    *nxt;                   /*      ->next node */
66      CONNECTOR           owner;                  /*      owner of viewport */
67      short               row, col;               /*      start of viewport */
68      short               height, width;          /*      extent */
69  } VIEW;
70
71  typedef struct appl_node                        /* links applications: */
72  {
73      struct appl_node    *nxt;                   /*      ->next node */
74      long                name;                   /*      name of application */
75      CONNECTOR           conn;                   /*      conn, to application */
76      short               row, col;               /*      origin */
77  } APPL;
78
79  typedef struct anim_node                        /* links animation processes: */
80  {
81      struct anim_node    *nxt;                   /*      ->next node */
82      long                name;                   /*      name of element */
83      CONNECTOR           conn;                   /*      conn. to process */
84  } ANIM;
85
86  typedef struct affected_area                    /* area changed by a request: */
87  {
88      short               r1, c1;                 /*      upper left front */
89      short               r2, c2;                 /*      lower right back */
90      char                color;                  /*      background color */
91      char                pattern;                /*      background pattern */
92      short               max_height;             /*      max. height */
93      short               max_width;              /*      max. width */
94      short               height, width;          /*      current size */
95  } AREA;
96
97  typedef struct lists                            /* list pointers, etc.: */
98  {
99      ELEMENT             *first;                 /*      ->pict. element list */
100     ELEMENT             *last;                  /*      ->end of p.e. list */
101     ELEMENT             *current;               /*      ->last p.e. changed */
102     VIEW                *views;                 /*      ->viewport list */
103     APPL                *appls;                 /*      ->applications list */
104     ANIM                *anims;                 /*      ->animation list */
105     int                 changes;                /*      #changes in list */
106     int                 erases;                 /*      #erasures in list */
107     int                 size;                   /*      #picture elements */
108     struct                                      /*      element pool descr.: */
109         {
110             long                n;              /*          #elements */
111             long                size;           /*          size of elements */
112             ELEMENT             *ptr;           /*          ->element buffer */
113         } pool;
114 } LIST;
115
116
```

```
117  /*  local functions  */
118
119  char        *value(), *tag();
120  ELEMENT     *mark_number(), *mark_area(), *mark_elements(), *new_element();
121  P_E_HDR     *first_macro(), *next_macro();
122
123  /* Picture manager:  main-line */
124
125  PROCESS(Picture)
126  {
127      CURRENT          cur;
128      AREA             area;
129      LIST             list;
130      register VIEW    *view;
131      register ANIM    *anim;
132
133      Set_event_key("Picture mgr.");
134      init_PM(&cur,&area,&list);
135      draw_picture(&cur,&area,&list);
136      for (view = list.views; view; view = view->nxt)
137          Put(DIRECT,view->owner,pid,Newmsg(32,"unmap",NULL));
138      for (anim = list.anims; anim; anim = anim->nxt)
139          Put(DIRECT,anim->conn.pid,Newmsg(32,"quit",NULL));
140      Exit();
141  }
142
143  init_PM(cur,area,list)
144  register CURRENT    *cur;
145  register AREA       *area;
146  register LIST       *list;
147  {
148      area->color = BLACK;
149      area->pattern = 0;
150      *cur->name = *cur->file = NULL;
151      area->max_height = area->max_width = 0;
152      list->current = list->first = list->last = NULL;
153      list->views = NULL;
154      list->appls = NULL;
155      list->anims = NULL;
156      list->size = list->pool.n = 0;
157      cur->debug = cur->check = cur->private = cur->display_mark = NO;
158      cur->mark = cur->old_mark = cur->erase_mark = NULL;
159  }
```

```
160
161  draw_picture(cur,area,list)
162  CURRENT         *cur;
163  register AREA   *area;
164  register LIST   *list;
165  {
166      register char       *msg;
167      register short      transaction = 0, result = 0, go = YES;
168      register ELEMENT    *element;
169      long                status[11], list_size = 0, *req = NULL;
170
171      while (go)
172      {
173          cur->msg = msg = Get(0,&cur->sender,&cur->size);
174          if (!transaction)
175          {
176              list->changes = list->erases = area->r2 = area->c2 = 0;
177              area->r1 = area->c1 = 32767;
178              cur->appl = NULL;
179              if (list->appls)
180                  check_appl(cur,list->appls);
181          }
182          if (*msg == '[' && transaction < 10)
183              status[++transaction] = 0;
184          else if (*msg == ']')
185              --transaction;
186          else
187              go = Request(cur,area,list,msg,cur->size,cur->appl);
188          if (!transaction)
189          {
190              if (list->changes)
191                  notify(cur,area,list);
192              for (element = list->first; element; element = element->nxt)
193              {
194                  element->changed = element->marked = NO;
195                  if (element->deleted && !Any_msg(NULL))
196                      delete_element(list,element);
197              }
198              if (Find_triple(msg,"rply",cur->size,NO,0,NULL) && result >= 0)
199                  reply_status(msg,msg,"completed",result);
200          }
201          free_requests(msg,cur->size,&req,&list_size);
202      }
203  }
```

```
204
205   check_appl(cur,appl)
206   register CURRENT     *cur;
207   register APPL        *appl;
208   {
209       for ( ; appl && (appl->conn.pid != cur->sender.pid); appl = appl->nxt);
210       if (appl)
211       {
212           if (!(cur->appl = appl->name))
213               cur->appl = -1;
214           cur->appl_row = appl->row;
215           cur->appl_col = appl->col;
216       }
217   }
218
219   free_requests(msg,size,req,list_size)
220   register char     *msg, **req;
221   register long     size, *list_size;
222   {
223       register char     *temp, *next;
224
225       if (msg)
226       {
227           *(char**)msg = *req;
228           *req = msg;
229           *list_size += size;
230           if (!Any_msg(NULL) || *list_size > 1000)
231               for (temp = *req, *req = NULL, *list_size = 0; temp; temp = next)
232               {
233                   next = *(char**)temp;
234                   Free(temp);
235               }
236       }
237   }
238
239   Request(cur,area,list,msg,size,appl)
240   register CURRENT     *cur;
241   register AREA        *area;
242   register LIST        *list;
243   register long        msg, size, appl;
244   {
```

```
245     register short    go = YES;
246
247     if (!strcmp(msg,"write"))
248         Draw(list,msg,size);
249     else if (!strcmp(msg,"edit"))
250         Edit_text(cur,area,list,msg,size,appl);
251     else if (!strcmp(msg,"mark"))
252         Move_mark(cur,area,list);
253     else if (!strcmp(msg,"hit"))
254         Hit(list,msg,size,appl);
255     else if (!strcmp(msg,"move"))
256         Move(area,list,msg,size,appl);
257     else if (!strcmp(msg,"erase"))
258         Erase(area,list,msg,size,appl);
259     else if (!strcmp(msg,"read"))
260         Copy(cur,area,list,msg,size,appl);
261     else if (!strcmp(msg,"replace"))
262         Replace(area,list,msg,size,appl);
263     else if (!strcmp(msg,"change"))
264         Change(area,list,msg,size,appl);
265     else if (!strcmp(msg,"animate"))
266         Animate(cur,list);
267     else if (!strcmp(msg,"alter") || !strcmp(msg,"cancel"))
268         Alter(cur,list);
269     else if (!strcmp(msg,"number"))
270         Query_number(list,msg,size,appl);
271     else if (!strcmp(msg,"mark?"))
272         Query_mark(cur);
273     else if (!strcmp(msg,"save"))
274         Save_picture(cur,list);
275     else if (!strcmp(msg,"set"))
276         Set_mark(cur,area,list);
277     else if (!strcmp(msg,"restore"))
278         Restore_mark(cur,area,list);
279     else if (!strcmp(msg,"bkgd"))
280         Background(area,list,msg,size);
281     else if (!strcmp(msg,"create"))
282         go = New_picture(cur,area,list);
283     else if (!strcmp(msg,"init"))
284         cur->private = go = New_picture(cur,area,list);
285     else if (!strcmp(msg,"open"))
286         go = Old_picture(cur,list);
287     else if (!strcmp(msg,"appl"))
288         Appl(cur,list);
289     else if (!strcmp(msg,"quit"))
290     {
291         if (go = (cur->sender.pid != cur->owner.pid))
292             reply_status(msg,msg,"not authorized",0);
293     }
```

```
294         else if (!strcmp(msg,"query"))
295             Query(cur,list);
296         else if (!strcmp(msg,"failed"))
297             Status(msg,size);
298         else if (!strcmp(msg,"done") || !strcmp(msg,"status"))
299             ;
300         else if(!Change_attribute(list,msg,size,appl))
301         {
302             if (!strcmp(msg,"view"))
303                 Viewport(cur,area,list);
304             else if (!strcmp(msg,"debug"))
305                 cur->debug = !cur->debug;
306             else
307                 reply_status(msg,"-\'unknown\'",msg,0);
308         }
309         return(go);
310     }
```

```
311
312  Change_attribute(list,msg,size,appl)
313  register LIST    *list;
314  register long    msg, size, appl;
315  {
316      static char          msgids[] = "select\0blink\0invert\0hide\0highlight\0";
317
318      register char        *p;
319      register short       new_state, changed, type;
320      register ELEMENT     *element;
321      register P_E_HDR     *hdr;
322
323      for (p = msgids, type = 0; *p && strcmp(msg,p); p += strlen(p)+1, ++type);
324      if (!*p)
325          return(NO);
326      list->current = element = mark_elements(list,NULL,NULL,msg,size,appl);
327      new_state = !(short)Find_triple(msg,"off ",size,NO,0,NULL);
328      for ( ; element; element = element->nxt)
329          if (element->marked)
330          {
331              hdr = (P_E_HDR *) &element->length;
332              switch (type)
333              {
334                  case 0:     changed = hdr->attr.selected || new_state;
335                              if (hdr->attr.selected = new_state)
336                                  Put(NEXT,"Console",Newmsg(hdr->length+50,
337                                      "write","data=#e#e; type=#c",hdr,NULL,'P'));
338                              break;
339                  case 1:     changed = hdr->attr.blink != new_state;
340                              hdr->attr.blink = new_state;
341                              break;
342                  case 2:     changed = hdr->attr.invert != new_state;
343                              hdr->attr.invert = new_state;
344                              break;
345                  case 3:     changed = hdr->attr.hidden != new_state;
346                              hdr->attr.hidden = new_state;
347                              break;
348                  case 4:     changed = hdr->attr.highlight != new_state;
349                              hdr->attr.highlight = new_state;
350              }
351              if (element->changed = changed)
352                  list->changes++;
353              element->marked = NO;
354          }
355      return(YES);
356  }
```

```
357
358   Query(cur,list)
359   CURRENT          *cur;
360   register LIST    *list;
361   {
362       unsigned             n_elem = 0, n_views = 0;
363       register unsigned    min_r = 65535, min_c = 65535;
364       register unsigned    max_r = 0, max_c = 0, pic_ht = 0, pic_wd = 0;
365       register ELEMENT     *element;
366       register P_E_HDR     *hdr;
367       register VIEW        *view;
368
369       for (element = list->first; element; element = element->nxt)
370       {
371           hdr = (P_E_HDR *) &element->length;
372           if (hdr->row < min_r) min_r = hdr->row;
373           if (hdr->col < min_c) min_c = hdr->col;
374           if (hdr->row + hdr=>height > max_r) max_r = hdr->row + hdr->height;
375           if (hdr->col + hdr->width > max_c) max_c = hdr->col + hdr->width;
376           n_elem++;
377       }
378       if (n_elem)
379       {
380           pic_ht = max_r - min_r;
381           pic_wd = max_c - min_c;
382       }
383       else
384           pic_ht = pic_wd = max_r = max_c = min_r = min_c = 0;
385       for (view = list=>views; view; view = view->nxt, n_views++);
386       Reply(cur->msg,
387           Newmsg(256,"status","orig=#S; size=#2s; low=#2s; high=#2s; cnt=#s; \
388           view=#s; name=#S; file=#S","picture",
389           pic_ht,pic_wd,min_r,min_c,max_r,max_c,n_elem,n_views,
390           cur=>name,cur->file));
391   }
392
393   Query_number(list,msg,size,appl)
394   register LIST        *list;
395   register long        msg, size, appl;
396   {
397       register unsigned    n = 0;
398       register ELEMENT     *element, *temp;
399
400       if (element = mark_elements(list,NULL,NULL,msg,size,appl))
401       {
402           for (temp = list->first; temp != element; temp = temp->nxt, n++);
403           Reply(msg,Newmsg(element->length+32,"number",
404                   "num=#s; elem=#e",n,&element->length));
405       }
406       else
407           reply_status(msg,"-number","too high",0);
408   }
```

```
409
410   Draw(list,msg,size)
411   register LIST    *list;
412   register long    msg, size;
413   {
414       register ELEMENT    *after;
415       register long       *p;
416
417       if (p = (long *) Find_triple(msg,"data",size,NULL,4,NULL))
418       {
419           if (Find_triple(msg,"back",size,NO,0,NULL))
420               after = NULL;
421           else
422               after = list->last;
423           if (!draw_elements(p,*(p-1),list,after))
424               reply_status(msg,"-write","bad length/type/macro",0);
425       }
426       else
427           reply_status(msg,"-write","missing \'data\'",0);
428   }
429
430   draw_elements(p,list_len,list,after)
431   register char       *p;
432   register long       list_len;
433   register LIST       *list;
434   register ELEMENT    *after;
435   {
436       register ELEMENT    *element;
437       register short      length, number = 0;
438
439       while ((length = *(short *) p)
440              && (list_len -= length) >= 0
441              && strchr("tlreacdsmn",((P_E_HDR*)p)->type))
442       {
443           if (((P_E_HDR*)p)->type == 'm' && !check_macro(p))
444               break;
445           element = new_element(list,length+sizeof(ELEMENT),after);
446           memcpy(&element->length,p,length);
447           if (!((P_E_HDR*)p)->height)
448               define_box(&element->length);
449           number++;
450           p += length;
451           Long_align(p);
452       }
453       list->size += number;
454       list->changes += number;
455       list->current = element;
456       return(length ? NO : YES);
457   }
```

```
458
459   define_box(hdr)
460   register P_E_HDR    *hdr;
461   {
462       register char   *val;
463
464       val = value(hdr);
465       if (hdr->type == 't')
466       {
467           hdr->height = VCHAR_HT;
468           hdr->width = VCHAR_WD * strlen(val+8);
469       }
470       else if ((hdr->type == 'n') || (hdr->type == 'm'))
471           ;
472   }
473
474   check_macro(hdr)
475   register P_E_HDR    *hdr;
476   {
477       register P_E_HDR    *temp, *first;
478       short               len;
479       char                *p, macro_type;
480
481       for (first = temp = first_macro(hdr,&macro_type,&len,&p); temp;
482           temp = next_macro(&len,&p))
483       {
484           if (macro_type == 'L')
485               temp->attr.hidden = YES;
486           if (!temp->height)
487               define_box(temp);
488       }
489       if (macro_type == 'L')
490           first->attr.hidden = NO;
491       return(p ? YES : NO);
492   }
```

```
493
494   P_E_HDR *first_macro(hdr,type,len,p)
495   register P_E_HDR    *hdr;
496   register char       *type;
497   register short      *len;
498   register char       **p;
499   {
500       register P_E_HDR    *temp;
501
502       *p = value(hdr);
503       if (type)
504           *type = **p;
505       (*p)++;
506       Long_align(*p);
507       temp = (P_E_HDR *) *p;
508       *len = hdr->length - (*p - (char *) hdr);
509       if (temp->length && temp->length < *len && strchr("tlreacdsmn",temp->type))
510           return(temp);
511       *p = NULL;
512       return(NULL);
513   }
514
515   P_E_HDR *next_macro(len,p)
516   register short      *len;
517   register char       **p;
518   {
519       register P_E_HDR    *temp;
520
521       if (*p)
522       {
523           temp = (P_E_HDR *) *p;
524           *p += temp->length;
525           Long_align(*p);
526           *len -= (*p - (char *) temp);
527           temp = (P_E_HDR *) *p;
528           if (temp->length)
529               if (temp->length < *len && strchr("tlreacdsmn",temp->type))
530                   return(temp);
531               else
532                   *p = NULL;
533       }
534       return(NULL);
535   }
```

```
536
537  Replace(area,list,msg,size,appl)
538  AREA              *area;
539  LIST              *list;
540  register long     msg, size, appl;
541  {
542      register char         *p;
543      register short        length = 0;
544      register ELEMENT      *temp;
545      register P_E_HDR      *hdr, *temp_hdr = NULL;
546      register long         list_len;
547      ELEMENT               *after = NULL;
548
549      if (Find_triple(msg,"@\0\0\0\0",size,NO,0,NULL))
550      {
551          Erase(area,list,msg,size,appl);
552          after = list->current;
553      }
554      if (p = Find_triple(msg,"data",size,NULL,1,NULL))
555      {
556          list_len = *((long *)(p-4));
557          while ((length = *(short *) p) && (list_len -= length) > 0)
558          {
559              hdr = (P_E_HDR *) p;
560              if (hdr->type == 'm' && !check_macro(hdr))
561                  break;
562              for (temp = list->last; temp &&
563                   ((P_E_HDR*)&temp->length)->row != hdr->row &&
564                   ((P_E_HDR*)&temp->length)->col != hdr->col; temp = temp->pre);
565              if (temp)
566              {
567                  temp_hdr = (P_E_HDR *) &temp->length;
568                  temp->deleted = YES;
569                  after = temp->pre;
570              }
571              draw_elements(hdr,length,list,after);
572              if (temp_hdr && (hdr->type != 't' || hdr->height != temp_hdr->height
573                  || hdr->width != temp_hdr->width))
574              {
575                  change_area(area,temp_hdr->row,temp_hdr->col,
576                              temp_hdr->height,temp_hdr->width);
577                  list->erases++;
578              }
579              p += length;
580              Long_align(p);
581          }
582      }
583      if (length)
584          reply_status(msg,"-replace","bad length/type/macro",0);
585  }
```

```
586
587   Erase(area,list,msg,size,appl)
588   AREA                *area;
589   register LIST       *list;
590   register long       msg, size, appl;
591   {
592       register ELEMENT    *element = NULL;
593       register P_E_HDR    *hdr;
594       int                 number;
595
596       if (element = mark_elements(list,NULL,&number,msg,size,appl))
597       {
598           list->current = element->pre;
599           for ( ; element; element = element->nxt)
600               if (element->marked)
601               {
602                   element->deleted = YES;
603                   hdr = (P_E_HDR*) &element->length;
604                   change_area(area,hdr->row,hdr->col,hdr->height,hdr->width);
605               }
606           list->erases += number;
607           list->changes += number;
608       }
609   }
610
611   Copy(cur,area,list,msg,size,appl)
612   CURRENT             *cur;
613   register AREA       *area;
614   register LIST       *list;
615   register long       msg, size, appl;
616   {
617       register ELEMENT    *element;
618       register short      bkgd, *p;
619       short               *q;
620       unsigned int        length = 0;
621
622       if (bkgd = (short) Find_triple(msg,"bkgd",size,NO,0,NULL))
623       {
624           p = (short *) Find_triple(msg,"@pos",size,&none,0,NULL);
625           q = (short *) Find_triple(msg,"@end",size,&none,0,NULL);
626           change_area(area,*p,*(p+1),*q-*p,*(q+1)-*(p+1));
627       }
628       if ((element = mark_elements(list,&length,NULL,msg,size,appl)) || bkgd)
629           send(cur,area,list,0,length,element,NO,NO,bkgd);
630       else
631           Reply(msg,Newmsg(64,"write",NULL));
632   }
```

```
633
634  Move(area,list,msg,size,appl)
635  AREA     *area;
636  LIST     *list;
637  long     msg, size, appl;
638  {
639      register ELEMENT     *element;
640      register P_E_HDR     *hdr;
641      register int         delta_row, delta_col, by_offset = NO, row = 0, col = 0;
642      register char        *p;
643      int                  n;
644
645      if (p = Find_triple(msg,"by  ",size,NULL,4,NULL))
646      {
647          by_offset = YES;
648          delta_row = *((short *) p)++;
649          delta_col = *(short *) p;
650      }
651      else if (p = Find_triple(msg,"to  ",size,NULL,4,NULL))
652      {
653          row = *((short *) p)++;
654          col = *(short *) p;
655      }
656      if (list->current = element = mark_elements(list,NULL,&n,msg,size,appl))
657      {
658          if (!by_offset)
659          {
660              hdr = (P_E_HDR *) &element->length;
661              delta_row = row - hdr->row;
662              delta_col = col - hdr->col;
663          }
664          for ( ; element; element = element->nxt)
665              if (element->marked)
666              {
667                  hdr = (P_E_HDR *) &element->length;
668                  change_area(area,hdr->row,hdr->col,hdr->height,hdr->width);
669                  hdr->row += delta_row;
670                  hdr->col += delta_col;
671                  element->changed = YES;
672                  element->marked = NO;
673                  element->deleted = (hdr->row < 0 || hdr->col < 0);
674              }
675          list->changes += n;
676          list->erases += n;
677      }
678  }
```

```
679
680   Change(area,list,msg,size,appl)
681   register AREA     *area;
682   register LIST     *list;
683   register long     msg, size, appl;
684   {
685       register ELEMENT    *element = NULL;
686       register P_E_HDR    *hdr;
687       char                *color, *bkgd, *fill, *pat;
688
689       color = Find_triple(msg,"colr",size,NULL,1,NULL);
690       bkgd = Find_triple(msg,"bkgd",size,NULL,1,NULL);
691       fill = Find_triple(msg,"fill",size,NULL,1,NULL);
692       pat = Find_triple(msg,"pat ",size,NULL,1,NULL);
693       if (list->current = element = mark_elements(list,NULL,NULL,msg,size,appl))
694           for ( ; element; element = element->nxt)
695               if (element->marked)
696               {
697                   hdr = (P_E_HDR*) &element->length;
698                   if (color)
699                       hdr->color = *color;
700                   if (bkgd)
701                       hdr->bkgrnd = *bkgd;
702                   if (fill)
703                       hdr->fill = *fill;
704                   if (pat)
705                       hdr->pattern = *pat;
706                   change_area(area,hdr->row,hdr->col,hdr->height,hdr->width);
707                   list->changes++;
708               }
709   }
710
711   Background(area,list,msg,size)
712   register AREA       *area;
713   register LIST       *list;
714   register long       msg, size;
715   {
716       area->color = *Find_triple(msg,"colr",size,&area->color,1,NULL);
717       area->pattern = *Find_triple(msg,"pat ",size,&area->pattern,1,NULL);
718       change_area(area,0,0,MAX_ROW,MAX_COL);
719       list->changes = list->erases = 1;
720   }
```

```
721
722   New_picture(cur,area,list)
723   register CURRENT    *cur;
724   register AREA       *area;
725   register LIST       *list;
726   {
727       register ELEMENT     *element;
728       register long        max, maxe;
729       short                def_max = 20, def_maxe = 100;
730       char                 def_bkgd = BLACK, def_pat = 0;
731
732       for (element = list->first; element; element = element->nxt)
733           element->deleted = YES;
734       list->current = list->first = list->last = NULL;
735       list->changes = list->erases = list->size = 0;
736       if (Find_triple(cur->msg,"file",cur->size,NO,0,NULL))
737           return(Old_picture(cur,list));
738       else
739       {
740           cur->owner = cur->sender;
741           strcpy(cur->name,Find_triple(cur->msg,"name",cur->size,&none,1,NULL));
742           area->max_height =
743               *(short*)Find_triple(cur->msg,"size",cur->size,&none,4,NULL);
744           area->max_width =
745               *(short *) (Find_triple(cur->msg,"size",cur->size,&none,4,NULL)+2);
746           area->color = *Find_triple(cur->msg,"bkgd",cur->size,&def_bkgd,1,NULL);
747           area->pattern = *Find_triple(cur->msg,"pat ",cur->size,&def_pat,1,NULL);
748           cur->highlight = *Find_triple(cur->msg,"high",cur->size,&none,1,NULL);
749           cur->check = (area->max_height != 0);
750           max =
751               (*(short*)Find_triple(cur->msg,"max ",cur->size,&def_max,2,NULL))+1;
752           maxe = *(short*)Find_triple(cur->msg,"maxe",cur->size,&def_maxe,2,NULL);
753           if (maxe & 1)
754               ++maxe;
755           list->pool.n = max;
756           list->pool.size = maxe + sizeof(ELEMENT) + 10;
757           list->pool.ptr = (ELEMENT *) Alloc(max*list->pool.size,YES);
758           memset(list->pool.ptr,0,max*list->pool.size);
759           change_area(area,0,0,MAX_ROW,MAX_COL);
760           list->changes = list->erases = 1;
761           reply_status(cur->msg,"+create","complete",0);
762           return(YES);
763       }
764   }
```

```
765
766   Old_picture(cur,list)
767   register CURRENT     *cur;
768   register LIST        *list;
769   {
770       register char    *p = (char*)1;
771       CONNECTOR        file;
772
773       strcpy(cur->name,Find_triple(cur->msg,"name",cur->size,&none,1,NULL));
774       strcpy(cur->file,Find_triple(cur->msg,"file",cur->size,cur->name,1,NULL));
775       if (*cur->file)
776       {
777           if (Connect_to(NEXT,"File mgt",Newmsg(64,"open",
778               "name=#S; omod=#S; amod=#S",cur->file,"R",NULL),&file))
779           {
780               cur->owner = cur->sender;
781               while (p)
782                   if (p = Call(DIRECT,file.pid,
783                       Newmsg(64,"read","conn=#C; size=#l",&file,-1),0,0))
784                           if (p = Find_triple(p,"data",0,NULL,4,NULL))
785                               draw_elements(p,*(long*)(p-4),list,NULL);
786               Put(DIRECT,file.pid,Newmsg(32,"close","conn=#C",&file));
787               reply_status(cur->msg,"+open","complete",0);
788               return(YES);
789           }
790           else
791               reply_status(cur->msg,"-open","can't open file",0);
792       }
793       else
794           reply_status(cur->msg,"-open","no file name",0);
795       return(NO);
796   }
```

```
797
798   Save_picture(cur,list)
799   CURRENT     *cur;
800   LIST        *list;
801   {
802       register char         *file_name, *m, *p;
803       register ELEMENT      *element;
804       CONNECTOR             file;
805       unsigned int          length = 0, num;
806
807       if (!(file_name = Find_triple(cur->msg,"file",cur->size,NULL,1,NULL)))
808           file_name = cur->file;
809       if (*file_name)
810           if (element =
811               mark_elements(list,&length,&num,cur->msg,cur->size,cur->appl))
812           {
813               if (!Connect_to(NEXT,"File_mgt",Newmsg(64,"open"
814                   "name=#S; omod=#S; amod=#S",file_name,"W",NULL),&file))
815                       Connect_to(NEXT,"File_mgt",Newmsg(64,"create",
816                           "name=#S; omod=#S; amod=#S",file_name,"W",NULL),&file);
817               if (file.pid)
818               {
819                   num = length + 4 * num + 4;
820                   m = Newmsg(num+50,"write","conn=#C; data=#A",&file,num,NULL);
821                   p = Find_triple(m,"data",0,NULL,1,NULL);
822                   for ( ; element; element = element->nxt)
823                       if (element->marked)
824                       {
825                           memcpy(p,element,element->length);
826                           p += element->length;
827                           Long_align(p);
828                       }
829                   *(short *) p = NULL;
830                   Put(DIRECT,file.pid,m);
831                   Put(DIRECT,file.pid,Newmsg(32,"close","conn=#C",&file));
832                   reply_status(cur->msg,"+save","picture saved",0);
833               }
834               else
835                   reply_status(cur->msg,"-save","can't open/create file",0);
836           }
837           else
838               reply_status(cur->msg,"-save","no elements",0);
839       else
840           reply_status(cur->msg,"-save","no file name",0);
841   }
```

```
842
843   Appl(cur,list)
844   CURRENT          *cur;
845   register LIST    *list;
846   {
847       register APPL    *appl;
848       register long    name;
849       register short   *p;
850
851       name = *(long *) Find_triple(cur->msg,"name",cur->size,&none,4,NULL);
852       for (appl = list->appls; appl && appl->name != name; appl = appl->nxt);
853       if (!appl)
854           appl = (APPL *) Alloc(sizeof(APPL),YES);
855       appl->conn = cur->sender;
856       p = (short *) Find_triple(cur->msg,"org ",cur->size,&none,2,NULL);
857       appl->row = *p++;
858       appl->col = *p;
859       appl->name = name;
860       appl->conn =
861           *(CONNECTOR *) Find_triple(cur->msg,"appl",cur->size,&none,4,NULL);
862       appl->nxt = list->appls;
863       list->appls = appl;
864   }
865
866   Move_mark(cur,area,list)
867   register CURRENT     *cur;
868   register AREA        *area;
869   register LIST        *list;
870   {
871       register P_E_HDR     *hdr;
872       register short       *pos;
873       char                 *q;
874
875       if (pos = (short *) Find_triple(cur->msg,"at  ",cur->size,NULL,4,NULL))
876       {
877           if (cur->mark)
878               erase_mark(cur,area);
879           else
880           {
881               q = cur->mark = Alloc(sizeof(P_E_HDR)+30,YES);
882               draw_line(&q,0,0,VCHAR_HT,0,NULL,YELLOW,'S',0,1,NULL);
883           }
884           hdr = (P_E_HDR *) cur->mark;
885           hdr->row = *pos++ - ((hdr->height - VCHAR_HT) / 2);
886           hdr->col = *pos;
887           cur->display_mark = YES;
888           list->changes++;
889       }
890   }
```

```
891
892     Query_mark(cur)
893     register CURRENT    *cur;
894     {
895         register P_E_HDR    *hdr;
896
897         if (hdr = (P_E_HDR *) cur->mark)
898             Reply(cur->msg,Newmsg(64,"mark","at=#2s",hdr->row,hdr->col));
899         else
900             reply_status(cur->msg,"-mark?","no mark defined",0);
901     }
902
903     Set_mark(cur,area,list)
904     register CURRENT    *cur;
905     register AREA       *area;
906     register LIST       *list;
907     {
908         register P_E_HDR    *hdr;
909
910         if ((hdr = (P_E_HDR*)Find_triple(cur->msg,"data",cur->size,NULL,1,NULL))
911             && hdr->length)
912         {
913             if (cur->mark)
914             {
915                 erase_mark(cur,area);
916                 Free(cur->mark);
917                 Free(cur->erase_mark);
918                 cur->erase_mark = NULL;
919             }
920             cur->mark = Alloc(hdr->length,YES);
921             memcpy(cur->mark,hdr,hdr->length);
922             cur->display_mark = YES;
923         }
924         else
925         {
926             if (cur->old_mark)
927                 Free(cur=>old_mark);
928             cur->old_mark = cur->mark;
929             cur->mark = NULL;
930         }
931         list->changes++;
932     }
```

```
933
934   Restore_mark(cur,area,list)
935   register CURRENT    *cur;
936   register AREA       *area;
937   register LIST       *list;
938   {
939       if (cur->old_mark)
940       {
941           if (cur->mark)
942           {
943               erase_mark(cur,area);
944               Free(cur->mark);
945               Free(cur->erase_mark);
946               cur->erase_mark = NULL;
947           }
948           cur->mark = cur->old_mark;
949           cur->old_mark = NULL;
950           list->changes++;
951       }
952   }
953
954
955   erase_mark(cur,area)
956   register CURRENT    *cur;
957   register AREA       *area;
958   {
959       if (!cur->erase_mark)
960           cur->erase_mark = Alloc(*(short*)cur->mark,YES);
961       memcpy(cur->erase_mark,cur->mark,*(short*)cur->mark);
962       ((P_E_HDR *)cur->erase_mark)->color = area->color;
963   }
964
965   Edit_text(cur,area,list,msg,size,appl)
966   CURRENT          *cur;
967   AREA             *area;
968   register LIST    *list;
969   register long    msg, size;
970   long             appl;
971   {
```

```
register char   *p, c, *text_start, *new;
register short  shift, offset, sel_offset, ok = YES;
short           sel_length;
ELEMENT         *element;
P_E_HDR         *hdr;

if (list->current = element = mark_elements(list,NULL,NULL,msg,size,appl))
{
    offset = *(short *) Find_once(msg,"offs",size,&none,2,NULL);
    hdr = (P_E_HDR *) &element->length;
    if (hdr->type == 't')
    {
        text_start = (p = value(hdr) + sizeof(long)) + 2 * sizeof(short);
        if (shift = *(short*)Find_once(msg,"shft",size,&none,2,0))
            shift_text(p,text_start,shift);
        if (Find_once(msg,"sel ",size,NO,0,NULL))
        {
            sel_offset = *((short *) p)++;
            sel_length = *(short *) p;
            ok = (offset < sel_length);
            offset += sel_offset;
        }
        if (ok = (ok && (offset < strlen(text_start))))
        {
            p = text_start + offset;
            if (new = Find_once(msg,"new ",size,NULL,1,NULL))
            {
                while (c = *new++)
                    if (c > 31 && c < 127 && *p)
                        *p++ = c;
                    else if (c == 8 && p > text_start)
                        *--p = ' ';
            }
            if (Find_once(msg,"blnk",size,NO,0,NULL))
                for ( ; *p; *p++ = ' ');
            if (Find_triple(msg,"by  ",size,NO,0,NULL))
            {
                Move(area,list,msg,size,appl);
                Draw(list,msg,size);
            }
            else
            {
                element->changed = YES;
                list->changes++;
            }
            Move_mark(cur,area,list);
            if (Find_once(msg,"fast",size,NO,0,NULL))
                list=>erases = 0;
```

```
1020                )
1021                else
1022                    reply_status(msg,"-edit","outside text string",0);
1023            }
1024            else
1025                reply_status(msg,"-edit","not a text element",0);
1026        }
1027        else
1028            reply_status(msg,"-edit","not found",0);
1029    }
1030
1031    shift_text(sel,text,nchars)
1032    register short   *sel, nchars;
1033    register char    *text;
1034    {
1035        register short   length, n;
1036
1037        if (length = strlen(text))
1038            if (nchars < 0 && (n = length + nchars) > 0)
1039            {
1040                memcpy(text,text+n,-nchars);
1041                memset(text-nchars,' ',n);
1042                if (*sel - n >= 0)
1043                    *sel -= n;
1044                else
1045                {
1046                    *sel = 0;
1047                    *(sel+1) += *sel - n;
1048                }
1049            }
1050            else if (nchars > 0 && (n = length - nchars) > 0)
1051            {
1052                memcpy(text+length,text+nchars,nchars);
1053                memset(text,' ',n);
1054                if (*sel + n < length)
1055                    *sel -= n;
1056                else
1057                {
1058                    *sel = length - n;
1059                    *(sel+1) -= *sel + n - length;
1060                }
1061            }
1062    }
```

```
1063
1064    Animate(cur,list)
1065    register CURRENT    *cur;
1066    register LIST       *list;
1067    {
1068        register ANIM   *anim;
1069        register char   *name;
1070        register long   pid;
1071        char            *m;
1072
1073        if (name = Find_triple(cur->msg,"name",cur->size,NULL,2,NULL))
1074            if (strlen(name) < 16)
1075            {
1076                for (anim = list->anims; anim && strcmp(name,anim->name);
1077                     anim = anim->nxt);
1078                if (!anim)
1079                    if (pid = NewProc(name,"//processes/animate",YES,-1))
1080                    {
1081                        anim = (ANIM *) Alloc(sizeof(ANIM),YES);
1082                        anim->conn.pid = pid;
1083                        strcpy(anim->name,name);
1084                        anim->nxt = list->anims;
1085                        list->anims = anim;
1086                        m = Alloc(cur->size,YES);
1087                        memcpy(m,cur->msg,cur->size);
1088                        Put(DIRECT,anim->conn.pid,m);
1089                    }
1090                    else
1091                        reply_status(cur->msg,"-animate","not supported",0);
1092                else
1093                    reply_status(cur->msg,"-animate","duplicate name",0);
1094            }
1095            else
1096                reply_status(cur->msg,"-animate","name too long",0);
1097    }
```

```
1098
1099    Alter(cur,list)
1100    register CURRENT    *cur;
1101    register LIST       *list;
1102    {
1103        register ANIM   *anim;
1104        register char   *name;
1105        CONNECTOR       conn;
1106
1107        if (name = Find_triple(cur->msg,"name",cur->size,NULL,2,NULL))
1108        {
1109            for (anim = list->anims; anim && strcmp(name,anim->name);
1110                 anim = anim->nxt);
1111            if (anim)
1112            {
1113                conn = anim->conn;
1114                if (!strcmp(cur->msg,"cancel"))
1115                {
1116                    list->anims = anim->nxt;
1117                    Free(anim);
1118                }
1119                Forward(DIRECT,conn.pid,cur->msg);
1120                cur->msg = NULL;
1121            }
1122            else
1123                reply_status(cur->msg,cur->msg,"not found",0);
1124        }
1125    }
```

```
1126
1127    Hit(list,msg,size,appl)
1128    register LIST    *list;
1129    register long    msg, size, appl;
1130    {
1131        register short      *p, tolerance;
1132        register ELEMENT    *element;
1133        register P_E_HDR    *hdr;
1134        ELEMENT             *find_box();
1135
1136        tolerance = *(short *) Find_triple(msg,"tolr",size,&none,2,NULL);
1137        if (p = (short *) Find_triple(msg,"pos ",size,&none,4,NULL))
1138            if (list->current = element = find_box(*p,*(p+1),list,appl))
1139            {
1140                hdr = (P_E_HDR *) &element->length;
1141                if (Find_triple(msg,"sel ",size,NO,0,NULL) && hdr->attr.selectable)
1142                {
1143                    hdr->attr.selected = YES;
1144                    if ((hdr->type == 'm') && (*value(hdr) == 'L'))
1145                        sel_list(hdr);
1146                    element=>changed = YES;
1147                    list->changes++;
1148                }
1149                Reply(msg,Newmsg(hdr->length+50,"write","data=#e#e",hdr,NULL));
1150            }
1151            else
1152                reply_status(msg,msg,"not found",0);
1153        else
1154            reply_status(msg,msg,"missing \'pos\'",0);
1155    }
```

```
1156
1157    ELEMENT *find_box(row,col,list,appl)
1158    register short  row, col;
1159    register LIST   *list;
1160    register long   appl;
1161    {
1162        register P_E_HDR    *hdr;
1163        register ELEMENT    *element;
1164
1165        for (element = list->last; element; element = element->pre)
1166        {
1167            hdr = (P_E_HDR *) &element->length;
1168            if (in_box(hdr->row,hdr->col,hdr->height,hdr->width,row,col)
1169                &&-!element->deleted)
1170                    if (!appl || (appl == -1 && !*(long*)(hdr+1))
1171                              || appl == *(long*)(hdr+1))
1172                        break;
1173        }
1174        return(element);
1175    }
1176
1177    in_box(r,c,h,w,c1,c2)
1178    register short  r, c, c1, c2, h, w;
1179    {
1180        if ((c1 < r) || (c2 < c))
1181            return(NO);
1182        if ((c1 > r + h) || (c2 > c + w))
1183            return(NO);
1184        return(YES);
1185    }
1186
1187    sel_list(hdr)
1188    register P_E_HDR    *hdr;
1189    {
1190        register P_E_HDR    *temp, *first;
1191        short               len;
1192        char                *p;
1193
1194        for (first = temp = first_macro(hdr,NULL,&len,&p);
1195            temp && temp->attr.hidden; temp = next_macro(&len,&p));
1196        if (temp)
1197        {
1198            temp->attr.hidden = YES;
1199            if (!(temp = next_macro(&len,&p)))
1200                temp = first;
1201            temp->attr.hidden = NO;
1202        }
1203    }
```

```
1204
1205    Viewport(cur,area,list)
1206    register CURRENT    *cur;
1207    register AREA       *area;
1208    register LIST       *list;
1209    {
1210        register VIEW   *view, *prev = NULL;
1211        CONNECTOR       *conn;
1212        ELEMENT         *element;
1213        unsigned int    length = 0;
1214        char            *p;
1215
1216        if (p = Find_triple(cur->msg,"area",cur->size,NULL,8,NULL))
1217        {
1218            for (view = list->views; view && (view->owner.pid != cur->sender.pid);
1219                 view = view->nxt) ;
1220            if (view)
1221                memcpy(&view->row,p,4*sizeof(short));
1222            else
1223            {
1224                view = (VIEW *) Alloc(sizeof(VIEW),YES);
1225                view->nxt = list->views;
1226                view->owner = cur->sender;
1227                memcpy(&view->row,p,4*sizeof(short));
1228                list->views = view;
1229            }
1230            change_area(area,view->row,view->col,view->height,view->width);
1231            element = mark_area(area->r1,area->c1,area->r2,area->c2,list,
1232                      MAX_P_E,NULL,NULL,NULL,&length,NULL,cur->appl);
1233            send(cur,area,list,0,length,element,YES,cur->display_mark,YES);
1234        }
1235        else
1236        {
1237            conn = (CONNECTOR *) Find_triple(cur->msg,"conn",0,&cur->sender,8,NULL);
1238            for (view = list->views; view && (view->owner.pid != conn->pid);
1239                 prev = view, view = view->nxt) ;
1240            if (view)
1241            {
1242                if (prev)
1243                    prev->nxt = view->nxt;
1244                else
1245                    list->views = view->nxt;
1246                Free(view);
1247            }
1248        }
1249    }
```

```
1250
1251    change_area(area,row,col,height,width)
1252    register AREA   *area;
1253    register short  row, col, height, width;
1254    {
1255        if (row < area->r1)
1256            area->r1 = row;
1257        if (col < area->c1)
1258            area->c1 = col;
1259        if (row + height > area->r2)
1260            area->r2 = row + height;
1261        if (col + width > area->c2)
1262            area->c2 = col + width;
1263    }
1264
1265    notify(cur,area,list)
1266    register CURRENT    *cur;
1267    register AREA       *area;
1268    LIST                *list;
1269    {
1270        register VIEW       *view;
1271        register int        length;
1272
1273        for (view = list->views; view; view = view->nxt)
1274        {
1275            length = mark_changes(list->first,
1276                view->row,view->col,view->height,view->width);
1277            send(cur,area,list,&view->owner,length,list->first,
1278                 YES,cur->display_mark,list->erases);
1279        }
1280    }
1281
1282    mark_changes(element,r,c,h,w)
1283    register ELEMENT    *element;
1284    register short      r, c, h, w;
1285    {
1286        register P_E_HDR    *hdr;
1287        register int        list_length = 0;
1288
1289        for ( ; element && !element->changed; element = element->nxt) ;
1290        for ( ; element; element = element->nxt)
1291        {
1292            hdr = (P_E_HDR *) &element->length;
1293            if (element->marked = (element->changed && !hdr->attr.hidden &&
1294                (hdr->row + hdr->height >= r) && (hdr->row <= r + h) &&
1295                (hdr->col + hdr->width >= c) && (hdr->col <= c + w)))
1296                    list_length += hdr->length + 3;
1297        }
1298        return(list_length);
1299    }
```

```
1300
1301    send(cur,area,list,proc,length,element,modify,mark,redraw)
1302    register CURRENT            *cur;
1303    AREA                        *area;
1304    LIST                        *list;
1305    register CONNECTOR          *proc;
1306    register unsigned int       length;
1307    register ELEMENT            *element;
1308    register unsigned short modify, mark, redraw;
1309    {
1310        register P_E_HDR        *hdr;
1311        register short          element_length;
1312        char                    *m, *p, *set_mark();
1313        ELEMENT                 *redraw_bkgd();
1314
1315        if (redraw)
1316            element = redraw_bkgd(area,list,&m,&p);
1317        else
1318            p = (m = Newmsg(length+300,
1319                    "write","data=#A; type=#c",length+250,NULL,'P')) + 24;
1320        if (element)
1321            for ( ; element; element = element->nxt)
1322            {
1323                if (element->marked && !element->deleted)
1324                {
1325                    element->marked = NO;
1326                    element_length = element->length;
1327                    memcpy(p,&element->length,(long)element_length);
1328                    hdr = (P_E_HDR *) p;
1329                    if (modify)
1330                    {
1331                        if (hdr->attr.selected)
1332                            element_length = set_select(hdr,cur->highlight);
1333                        if ((hdr->type == 'm') && (*value(hdr) == 'L'))
1334                            element_length = macro_list(hdr);
1335                        if ((hdr->type == 't'))
1336                            element_length = check_text(hdr,hdr->length);
1337                        if (cur->appl)
1338                            element_length =
1339                                change_origin(hdr,cur->appl_row,cur->appl_col);
1340                    }
1341                    p += element_length;
1342                    Long_align(p);
1343                }
1344            }
1345        if (mark)
1346            p = set_mark(p,cur);
1347        *(short *) p = NULL;
1348        if (proc)
1349            Put(DIRECT,proc->pid,m);
1350        else
1351            Reply(cur->msg,m);
1352    }
```

```
1353
1354    change_origin(hdr,row,col)
1355    register P_E_HDR    *hdr;
1356    register short      row, col;
1357    {
1358        if ((hdr->row -= row) < 0)
1359            return(0);
1360        if ((hdr->col -= col) < 0)
1361            return(0);
1362        return(hdr->length);
1363    }
1364
1365    char *set_mark(p,cur)
1366    register char       *p;
1367    register CURRENT    *cur;
1368    {
1369        if (cur->erase_mark)
1370        {
1371            memcpy(p,cur->erase_mark,*(short*)cur->erase_mark);
1372            p += *(short *) p;
1373        }
1374        if (cur->mark)
1375        {
1376            memcpy(p,cur->mark,*(short*)cur->mark);
1377            p += *(short *) p;
1378        }
1379        return(p);
1380    }
1381
1382    ELEMENT *redraw_bkgd(area,list,buf,ptr)
1383    register AREA   *area;
1384    register LIST   *list;
1385    register char   **buf, **ptr;
1386    {
1387        ELEMENT     *element;
1388        int         length, num;
1389
1390        element = mark_area(area->r1,area->c1,area->r2,area->c2,
1391                    list,MAX_P_E,NULL,NULL,NULL,&length,&num,NULL);
1392        length += (4 * num) + 150;
1393        *buf = Newmsg(length+50,"write","data=#A; type=#c",length,NULL,'P');
1394        *ptr = *buf + 24;
1395        draw_filled_rect(ptr,area->r1,area->c1,(area->r2)-(area->r1),
1396            (area->c2)-(area->c1),NULL,0,0,area->color,area->pattern,0,0,0,NULL);
1397        return(element);
1398    }
```

```
1399
1400    set_select(hdr,high_option)
1401    register P_E_HDR    *hdr;
1402    register char       high_option;
1403    {
1404        register short  length;
1405
1406        length = hdr->length;
1407        if (!high_option)
1408            hdr->attr.invert = !hdr->attr.invert;
1409        else if (high_option == 'I')
1410            hdr->attr.invert = !hdr->attr.invert;
1411        else if (high_option == 'H')
1412            hdr->attr.highlight = !hdr->attr.highlight;
1413        else if (high_option == 'C')
1414        {
1415            if (hdr->type != 'm')
1416            {
1417                hdr->color = (hdr->color + 1) % 7 + 1;
1418                if (hdr->fill)
1419                    hdr->fill = (hdr->fill + 1) % 7 + 1;
1420            }
1421            else
1422                macro_color(hdr);
1423        }
1424        else if (hdr->type == 't')
1425            sel_text(hdr,high_option);
1426        return(length);
1427    }
1428
1429    macro_list(hdr)
1430    register P_E_HDR    *hdr;
1431    {
1432        register P_E_HDR    *temp;
1433        register short      row, col;
1434        short               len;
1435        char                *p;
1436
1437        row = hdr->row;
1438        col = hdr->col;
1439        for (temp = first_macro(hdr,NULL,&len,&p);
1440            temp && temp->attr.hidden; temp = next_macro(&len,&p));
1441        if (temp)
1442        {
1443            memcpy(hdr,temp,temp->length);
1444            hdr->row = row;
1445            hdr->col = col;
1446        }
1447        return(hdr->length);
1448    }
```

```
1449
1450    macro_color(hdr)
1451    register P_E_HDR    *hdr;
1452    {
1453        register P_E_HDR    *temp;
1454        short               len;
1455        char                *p;
1456
1457        for (temp = first_macro(hdr,NULL,&len,&p); temp; temp = next_macro(&len,&p))
1458        {
1459            temp->color = (temp->color + 1) % 7 + 1;
1460            if (temp->fill)
1461                temp->fill = (temp->fill + 1) % 7 + 1;
1462        }
1463    }
1464
1465    sel_text(hdr,high_option)
1466    register P_E_HDR    *hdr;
1467    register char       high_option;
1468    {
1469        register TEXT_OPTIONS   *opt;
1470
1471        opt = (TEXT_OPTIONS *) value(hdr);
1472        if (high_option == 'b')
1473            opt->border = YES;
1474        else if (high_option == 'U')
1475            opt->underline = YES;
1476        else if (high_option == 'B')
1477            opt->bold = YES;
1478    }
1479
1480    check_text(hdr,length)
1481    register P_E_HDR    *hdr;
1482    register short      length;
1483    {
1484        register char           *p;
1485        char                    *n;
1486        register TEXT_OPTIONS   *opt;
1487
1488        opt = (TEXT_OPTIONS *) value(hdr);
1489        if (opt->border && hdr->fill)
1490        {
1491            opt->border = NO;
1492            p = (char *) hdr + length;
1493            Long_align(p);
1494            n = p;
1495            draw_rect(&n,hdr->row-3,hdr->col-3,hdr->height+6,hdr->width+6,
1496                NULL,hdr->fill,'S',1,NULL);
1497            length = n - (char*)hdr;
1498        }
1499        return(length);
1500    }
```

```
1501
1502    ELEMENT *mark_elements(list,length,num,msg,size,appl)
1503    LIST                    *list;
1504    unsigned int            *length, *num;
1505    long                    msg, size, appl;
1506    {
1507        register short      row = 0, col = 0, number = 0, count = 1;
1508        register short      to_row = MAX_ROW, to_col = MAX_COL, *p;
1509        register ELEMENT    *element;
1510        register char       *tag_pat = NULL;
1511        char                what = NULL, tag_buf[200], *text_pat = NULL, dflt = YES;
1512        long                *triple, attr = NULL;
1513
1514        element = NULL;
1515        while (p = (short*)Find_triple(msg,"@\0\0\0",size,NULL,0,&triple))
1516        {
1517            switch (*triple)
1518            {
1519                case Keypack('@','c','n','t'):   count = *p;
1520                                                 break;
1521                case Keypack('@','a','t','t'):   attr |= *(long *) p;
1522                                                 break;
1523                case Keypack('@','s','e','l'):   attr |= 0x8000;
1524                                                 break;
1525                case Keypack('@','n','u','m'):   number = *p;
1526                                                 what = NULL;
1527                                                 break;
1528                case Keypack('@','p','o','s'):   row = *p++;
1529                                                 col = *p;
1530                                                 what = 'A';
1531                                                 break;
1532                case Keypack('@','e','n','d'):   to_row = *p++;
1533                                                 to_col = *p;
1534                                                 what = 'A';
1535                                                 break;
1536                case Keypack('@','t','x','t'):   text_pat = Alloc(500,YES);
1537                                                 if (!makpat(p,text_pat))
1538                                                 {
1539                                                    Free(text_pat);
1540                                                    text_pat = NULL;
1541                                                 }
1542                                                 break;
1543                case Keypack('@','t','a','g'):   if (!makpat(p,(tag_pat = tag_buf)))
1544                                                    tag_pat = NULL;
1545            }
1546            *triple = NULL;
1547            dflt = NO;
1548        }
1549        if (dflt)
1550            count = MAX_P_E;
1551        if (!what)
1552            element = mark_number(number,tag_pat,text_pat,
1553                        list,count,attr,length,num,appl);
1554        else if (what == 'A')
1555            element = mark_area(row,col,to_row,to_col,list,count,
1556                        attr,tag_pat,text_pat,length,num,appl);
```

```
1557        if (text_pat)
1558            Free(text_pat);
1559        return(element);
1560    }
1561
1562    ELEMENT *mark_area(row,col,to_row,to_col,list,
1563                       count,attr,tag_pat,text_pat,length,num,appl)
1564    register short    row, col, to_row, to_col, count;
1565    long              attr;
1566    LIST              *list;
1567    char              *tag_pat, *text_pat;
1568    unsigned int      *length, *num;
1569    {
1570        register P_E_HDR    *hdr;
1571        register ELEMENT    *element = NULL, *temp;
1572        register long       total_length = 0;
1573        unsigned int        orig_count;
1574
1575        if (row >= 0 && col >= 0 && to_row >= 0 && to_col >= 0)
1576        {
1577            orig_count = count;
1578            for (temp = list->first; temp && count; temp = temp->nxt)
1579            {
1580                hdr = (P_E_HDR *) &temp->length;
1581                if (hdr->row <= to_row && hdr->col <= to_col
1582                    && row < hdr->row + hdr->height && col < hdr->col + hdr->width
1583                    && valid(hdr,tag_pat,text_pat,attr,appl) && !temp->deleted)
1584                    {
1585                        total_length += temp->length;
1586                        temp->marked = YES;
1587                        if (!element)
1588                            element = temp;
1589                        count--;
1590                    }
1591            }
1592            if (length)
1593                *length = total_length;
1594            if (num)
1595                *num = orig_count - count;
1596        }
1597        return(element);
1598    }
```

```
1599
1600    ELEMENT *mark_number(n,tag_pat,text_pat,list,count,attr,length,num,appl)
1601    register short    n, count;
1602    register long     tag_pat, text_pat, attr;
1603    register LIST     *list;
1604    unsigned int      *length, *num;
1605    {
1606        register ELEMENT    *element = NULL, *temp = NULL;
1607        register long       total_length = 0;
1608        unsigned int        orig_count;
1609
1610        if (n == -1)
1611            temp = list->last;
1612        else
1613            for (temp = list->first; temp && n--; temp = temp->nxt);
1614        for (orig_count = count; temp && count; temp = temp->nxt)
1615            if (valid(&temp->length,tag_pat,text_pat,attr,appl) && !temp->deleted)
1616            {
1617                total_length += temp->length;
1618                temp->marked = YES;
1619                if (!element)
1620                    element = temp;
1621                count--;
1622            }
1623        if (length)
1624            *length = total_length;
1625        if (num)
1626            *num = orig_count - count;
1627        return(element);
1628    }
```

```
1629
1630    valid(hdr,tag_pat,text_pat,attr,appl)
1631    register P_E_HDR    *hdr;
1632    register char       *tag_pat, *text_pat;
1633    register long       attr, appl;
1634    {
1635        register char    *target, ok = YES;
1636        long             temp;
1637
1638        if (tag_pat)
1639            if (target = tag(hdr))
1640                ok = amatch(target,tag_pat);
1641            else
1642                ok = NO;
1643        if (text_pat)
1644            if (hdr->type == 't')
1645                ok = ok && amatch(value(hdr)+8,text_pat);
1646            else
1647                ok = NO;
1648        if (attr)
1649        {
1650            memcpy(&temp,&hdr->attr,sizeof(long));
1651            ok = ok && (temp & attr);
1652        }
1653        if (appl)
1654        {
1655            if (appl == -1)
1656                ok = ok && (!*(long*)(hdr+1));
1657            else
1658                ok = ok && (appl == *(long*)(hdr+1));
1659        }
1660        return(ok);
1661    }
1662
1663    Status(msg,size)
1664    register char   *msg;
1665    register long   size;
1666    {
1667        register char   *m;
1668
1669        *(m = Alloc(size,YES)) = NULL;
1670        strcat(m,Find_triple(msg,"orig",size,&none,1,NULL));
1671        strcat(m," ");
1672        strcat(m,Find_triple(msg,"stat",size,&none,1,NULL));
1673        strcat(m," in-");
1674        strcat(m,Find_triple(msg,"req ",size,&none,1,NULL));
1675        Note(m,"ERROR");
1676        Free(m);
1677    }
```

```
1678
1679    reply_status(cur,mid,stat,code)
1680    register char    *cur, *mid, *stat;
1681    register long    *code;
1682    {
1683        register char   *type;
1684
1685        type = "failed";
1686        if (*mid == '-')
1687            mid++;
1688        else if (*mid == '+')
1689        {
1690            type = "done";
1691            mid++;
1692        }
1693        Reply(cur,Newmsg(strlen(mid)+strlen(stat)+50,type,
1694            "orig=#S; req=#S; stat=#S; code=#l","picture",mid,stat,code));
1695    }
1696
1697    ELEMENT *new_element(list,size,after)
1698    register LIST       *list;
1699    register long       size;
1700    register ELEMENT    *after;
1701    {
1702        register ELEMENT    *element;
1703        register long       i = 0;
1704
1705        if (size <= list->pool.size)
1706            for (i = list->pool.n, element = list->pool.ptr;
1707                 element->pool && i && !element->deleted;
1708                 (char*)element += list->pool.size, --i) ;
1709        if (i)
1710        {
1711            if (element->deleted)
1712                delete_element(list,element);
1713            element->pool = YES;
1714        }
1715        else
1716        {
1717            element = (ELEMENT *) Alloc(size,YES);
1718            element->pool = NO;
1719        }
1720        element->nxt = NULL;
1721        if (element->pre = list->last)
1722            (list->last)->nxt = element;
1723        else
1724            list->first = element;
1725        list->last = element;
1726        element->changed = YES;
1727        element->deleted = element->marked = NO;
1728        return(element);
1729    }
```

```
1730
1731    delete_element(list,element)
1732    register ELEMENT    *element;
1733    register LIST       *list;
1734    {
1735        if (element->pre)
1736            (element->pre)->nxt = element->nxt;
1737        else
1738            list->first = element->nxt;
1739        if (element->nxt)
1740            (element->nxt)->pre = element->pre;
1741        else
1742            list->last = element->pre;
1743        if (element->pool)
1744            element->pool = NULL;
1745        else
1746            Free(element);
1747        --list->size;
1748    }
1749
1750    char *value(hdr)
1751    register P_E_HDR    *hdr;
1752    {
1753        register char   *p;
1754
1755        p = (char *) hdr + sizeof(P_E_HDR);
1756        if (hdr->attr.appl)
1757            p += 4;
1758        if (hdr->attr.tagged)
1759            while (*p++);
1760        Long_align(p);
1761        return(p);
1762    }
1763
1764
1765
1766    char *tag(hdr)
1767    register P_E_HDR    *hdr;
1768    {
1769        register char   *p;
1770
1771        p = (char *) hdr + sizeof(P_E_HDR);
1772        if (hdr->attr.appl)
1773            p += 4;
1774        if (hdr->attr.tagged)
1775            return(p);
1776        return(NULL);
1777    }
```

What is claimed is:

1. A virtual input interface in a data processing system, said interface comprising:

means for accepting input from at least one physical device and for converting said physical device input into virtual input, said means comprising a virtual input manager process responsive to said at least one physical input device for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element; and

means responsive to said virtual input for performing processing operations upon said virtual input, said means comprising a console manager process for performing processing operations on said one or more picture elements.

2. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of keystrokes.

3. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of cursor position.

4. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of system-defined actions.

5. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of user-defined functions.

6. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of menu selections.

7. The virtual input interface as recited in claim 1, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

8. The virtual input interface as recited in claim 1, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

9. A virtual output interface in a data processing system, said interface comprising:

a source of virtual input, said virtual input comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said virtual input and for generating virtual output;

means for accepting said virtual output; and

means for converting said virtual output into at least one physical output suitable for use by at least one physical output device.

10. The virtual output interface as recited in claim 9, wherein said virtual input comprises a plurality of related picture elements and wherein said virtual output accepting means comprises a picture manager process for controlling said plurality of related picture elements.

11. The virtual output interface as recited in claim 10 and further comprising a display device, wherein said virtual output accepting means further comprises a window manager process for controlling the display of said plurality of related picture elements on said display device.

12. The virtual output interface as recited in claim 9, wherein said virtual output converting means comprises a virtual output manager process responsive to said one or more processed picture elements for coupling said one or more processed picture elements to said at least one physical output device.

13. The virtual output interface as recited in claim 9, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

14. The virtual output interface as recited in claim 9, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

15. In a data processing system, an interface between processes and data in said system and physical input and output devices coupled to said system, said interface comprising:

means responsive to one of said physical input devices for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said one or more picture elements; and

means responsive to said one or more processed picture elements for coupling said one or more processed picture elements to one of said physical output devices.

16. The data processing system as recited in claim 15, wherein said one or more picture elements define a graphical object and at least one attribute thereof.

17. The data processing system as recited in claim 16, wherein one of said data fields describes the length of the associated picture element.

18. The data processing system as recited in claim 16, wherein one of said data fields identifies the particular type of the associated picture element.

19. The data processing system as recited in claim 16, wherein one of said data fields describes the position of the associated picture element relative to row and column coordinates on a picture of which said picture element forms a part.

20. The data processing system as recited in claim 16, wherein one of said data fields describes the size of the associated picture element.

21. The data processing system as recited in claim 16, wherein one of said data fields describes the color of the associated picture element.

22. The data processing system as recited in claim 15, wherein said means responsive to one of said physical input devices comprises a virtual input manager process.

23. The data processing system as recited in claim 15, wherein said means responsive to said one or more processed picture elements comprises a virtual output manager process.

* * * * *