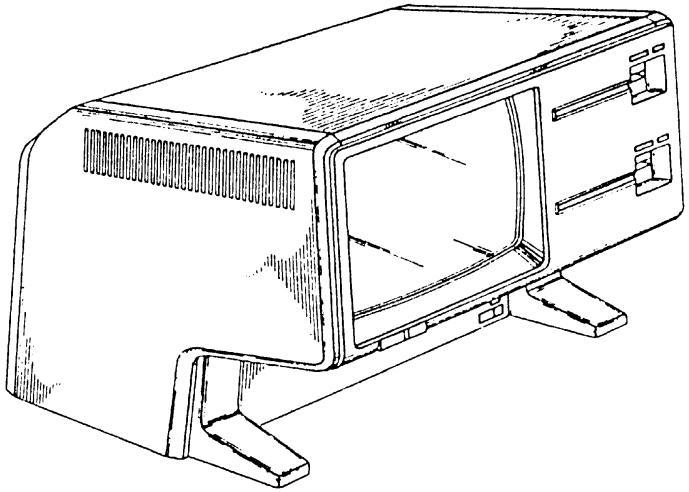
Apple Lisa ToolKit 3.0

Source Code Listing



This is a listing of all the Lisa ToolKit 3.0 source code files. These files are for the most part written in Lisa Clascal, an object-oriented Pascal that Apple Computer created from its Lisa Pascal compiler. Other source code languages exist here too which includes the Lisa Workshop EXEC language (the BUILD files are in this) and various 68000 assembly language sources such as LIBPL/CLASLIB.TEXT.

For detailed information about the Lisa ToolKit see Apple's extensive ToolKit documentation which includes the Lisa ToolKit Reference Manual, a Clascal primer, and a ToolKit tutorial.Tmagazine's 1984 article "Software Frameworks" which describes the ToolKit architecture and its various core classes.

```
================================================================================================
FILE: " ToolKit 3.0 Source Catalog"
================================================================================================

000001
000002             APPLE LISA TOOLKIT 3.0 SOURCE CODE DISK CATALOG LISTINGS
000003             ========================================================
000004
000005  This document contains catalog listings of the 4 disks which held all the
000006  source code files for the Lisa ToolKit 3.0 object library.
000007
000008  Document prepared by David T. Craig -- March 1993
000009
000010  +-------------------------------------------------------------------------
000011  |  TOOLKIT SOURCE DISK # 1
000012  +-------------------------------------------------------------------------
000013
000014  Filename                     Size Psize   Last-Mod-Date   Creation-Date  Attr
000015  --------                     ---- -----   -------------   -------------  ----
000016  libpl/CLASLIB.TEXT          12288    24  02/06/84-11:01  02/06/84-11:01
000017  libpl/UCLASCAL.TEXT         45056    88  08/29/84-14:49  04/02/84-16:44
000018  libtk/UABC.TEXT             69632   136  08/29/84-15:08  04/26/84-12:02  C
000019  libtk/UABC2.TEXT            93184   182  08/17/84-11:25  05/18/84-19:28  C
000020  libtk/UABC3.TEXT            66560   130  08/17/84-11:27  05/07/84-17:57
000021  LIBTK/UABC4.TEXT            61440   120  08/17/84-11:29  05/07/84-18:04
000022
000023  680 total blocks for files listed
000024  28 blocks of OS overhead for volume and files listed
000025  76 blocks free out of 772
000026
000027  +-------------------------------------------------------------------------
000028  |  TOOLKIT SOURCE DISK # 2
000029  +-------------------------------------------------------------------------
000030
000031  Filename                     Size Psize   Last-Mod-Date   Creation-Date  Attr
000032  --------                     ---- -----   -------------   -------------  ----
000033  LIBTK/UABC5.TEXT            95232   186  08/17/84-11:32  05/07/84-18:03
000034  libtk/UDIALOG.TEXT          44032    86  08/17/84-15:20  04/26/84-13:23
000035  libtk/UDIALOG2.TEXT         78848   154  08/17/84-15:29  04/25/84-16:28
000036  libtk/UDIALOG3.TEXT         61440   120  08/17/84-15:23  04/25/84-18:01  C
000037  libtk/UDIALOG4.TEXT         37888    74  04/25/84-18:58  04/25/84-18:58
000038  libtk/UDRAW.TEXT            22528    44  08/29/84-15:06  05/01/84-15:07
000039
000040  664 total blocks for files listed
000041  28 blocks of OS overhead for volume and files listed
000042  92 blocks free out of 772
000043
000044  +-------------------------------------------------------------------------
```

```
000045  |  TOOLKIT SOURCE DISK # 3
000046  +------------------------------------------------------------------------
000047
000048  Filename                     Size Psize   Last-Mod-Date   Creation-Date  Attr
000049  --------                     ---- -----   -------------   -------------  ----
000050  libtk/Udraw2.TEXT           54272   106  08/16/84-19:13  05/07/84-18:02  C
000051  LIBTK/UOBJECT.TEXT          43008    84  08/29/84-14:57  05/16/84-08:57
000052  libtk/UOBJECT2.TEXT         55296   108  08/17/84-14:01  05/01/84-15:50  C
000053  libtk/UOBJECT3.TEXT         70656   138  08/17/84-14:03  04/30/84-13:17
000054  libtk/UOBJECT4.TEXT         76800   150  08/17/84-14:07  04/30/84-14:26
000055  libtk/utext.text            33792    66  08/17/84-15:53  04/26/84-13:03  C
000056
000057  652 total blocks for files listed
000058  28 blocks of OS overhead for volume and files listed
000059  104 blocks free out of 772
000060
000061  +------------------------------------------------------------------------
000062  |  TOOLKIT SOURCE DISK # 4
000063  +------------------------------------------------------------------------
000064
000065  Filename                     Size Psize   Last-Mod-Date   Creation-Date  Attr
000066  --------                     ---- -----   -------------   -------------  ----
000067  BUILD/ASSEMB.TEXT            2048     4  12/12/83-18:55  12/12/83-18:55
000068  BUILD/COMP.TEXT              2048     4  08/16/84-13:20  09/21/83-14:49
000069  BUILD/INSTALL.TEXT           2048     4  08/16/84-13:20  05/15/84-14:58
000070  BUILD/MAKE/ATKLIB.TEXT       2048     4  02/02/84-15:47  02/02/84-15:47
000071  build/make/Ctk2lib.text      2048     4  02/24/84-15:50  02/24/84-15:50
000072  BUILD/MAKE/CTKLIB.TEXT       2048     4  08/16/84-13:20  02/24/84-15:50
000073  build/make/Ltk2lib.text      2048     4  08/16/84-13:21  04/26/84-11:31
000074  BUILD/MAKE/LTKLIB.TEXT       2048     4  08/16/84-13:21  05/01/84-15:23
000075  BUILD/MAKE/TKLIB.TEXT        2048     4  08/27/84-11:06  02/24/84-15:51
000076  INTERFACE/PASLIBC...TEXT     3072     6  11/13/85-13:56  11/13/85-13:18
000077  INTERFACE/PASSWD.TEXT        2048     4  11/13/85-13:17  11/13/85-13:17
000078  INTERFACE/PPASLIBC.TEXT      3072     6  11/13/85-13:59  11/13/85-13:18
000079  LIBPL/PASLIBCALL.OBJ         2560     5  04/04/84-14:07  04/04/84-14:07
000080  LIBPL/PPASLIBC.OBJ           2560     5  04/04/84-14:08  04/04/84-14:08
000081  libtk/passwd.OBJ             1536     3  08/16/84-13:47  08/16/84-13:47
000082  libtk/UTEXT2.TEXT           58368   114  08/17/84-15:55  04/25/84-17:01
000083  libtk/UTEXT3.TEXT           63488   124  08/17/84-15:56  05/21/84-09:25
000084  LIBTK/UTEXT4.TEXT          102400   200  08/17/84-15:58  05/21/84-09:30
000085  LIBTK/UUNIVTEXT.TEXT        13312    26  08/29/84-17:12  05/18/84-15:29
000086  libtk/XFER.TEXT             10240    20  04/25/84-20:36  04/25/84-20:36
000087  LIBUT/UUNIVTEXT2.TEXT       71680   140  05/23/84-09:49  05/18/84-17:44
000088  UFIXUTEXT.TEXT              12288    24  08/15/84-13:10  08/14/84-17:23
000089
000090  713 total blocks for files listed
000091  44 blocks of OS overhead for volume and files listed
000092  27 blocks free out of 772
```

```
000093
000094  THAT'S ALL, FOLKS !
000095

End of File -- Lines: 95 Characters: 4680
```

```
================================================================================================
FILE: "BUILD/ASSEMB.TEXT"
================================================================================================

000001  $EXEC   {Assemble a module }                {filename build/assemb.text}
000002  $
000003  ${ %0 -- pathname of the module to assemble}
000004  ${ %1 -- (optional) pathname of the resulting object file.  Default name is %0}
000005  ${ %2 -- (optional) segment name for the resulting object file.  Default is 'blank' segment}
000006  $
000007  $IF %0='' THEN
000008     $WRITE 'File To Assemble? '
000009     $READLN %0
000010     $IF %1='' THEN
000011        $WRITE "Name For Object File [<cr> For %0]? "
000012        $READLN %1
000013        $IF %2='' THEN
000014           $WRITE 'Segment Name [<cr> For Blank Segment]? '
000015           $READLN %2
000016        $ENDIF
000017     $ENDIF
000018  $ENDIF
000019  $DEFAULT %1 to %0
000020  A{ssemble}%0             {source file}
000021                           {no listing file}
000022  %1                       {object file}
000023  $IF %2<>'' THEN
000024     R{un}changeseg        {re-assign segmentation (optional)}
000025     %1
000026     y%2
000027  $ENDIF
000028  $ENDEXEC
000029

End of File -- Lines: 29 Characters: 744
```

```
================================================================================================
FILE: "BUILD/COMP.TEXT"
================================================================================================

000001  $EXEC     {Compile and Code Generate a Pascal Unit}    {filename build/comp.text}
000002  $
000003  ${ %0 -- pathname of the unit to compile}
000004  ${ %1 -- (optional) pathname of the resulting object file. Defaults to %0}
000005  ${        Destroys file 'temp/c.i'}
000006  ${ %2 -- (optional) pathname of intrinsic.lib.  Defaults to -#boot-intrinsic.lib}
000007  $
000008  $IF %0='' THEN
000009     $WRITE 'File To Compile? '
000010     $READLN %0
000011     $IF %1='' THEN
000012        $WRITE "Name For Object File [<cr> For %0]? "
000013        $READLN %1
000014        $IF %2='' THEN
000015           $WRITE 'Name Of Intrinsic.lib [<cr> For -#boot-intrinsic.lib]? '
000016           $READLN %2
000017        $ENDIF
000018     $ENDIF
000019  $ENDIF
000020  $DEFAULT %1 TO %0
000021  $DEFAULT %2 TO '-#boot-intrinsic.lib'
000022  P{ascal Compile}?{option flag}
000023  %2                                      {intrinsic.lib}
000024  %0                                      {source file}
000025                                          {no listing file}
000026  %1                                      {object file}
000027  $ENDEXEC
000028

End of File -- Lines: 28 Characters: 818
```

```
================================================================================================
FILE: "BUILD/INSTALL.TEXT"
================================================================================================

000001  $EXEC     {Install a Library in Intrinsic.lib}    {filename build/install.text}
000002  $
000003  ${ %0 -- number of the library to install}
000004  ${ %1 -- (optional)pathname for input intrinsic.lib.  Defaults to }
000005  ${        -#boot-intrinsic.lib}
000006  ${ %2 -- (optional) pathname for output intrinsic.lib.  Defaults to %1}
000007  $
000008  $IF %0='' THEN
000009     $WRITE 'Number Of The Library To Install? '
000010     $READLN %0
000011     $IF %1='' THEN
000012        $WRITE 'Pathname For Input Intrinsic.lib [<cr> For -#boot-intrinsic.lib]? '
000013        $READLN %1
000014        $IF %2='' THEN
000015           $WRITE 'Pathname For Output Intrinsic.lib [<cr> For -#boot-intrinsic.lib]? '
000016           $READLN %2
000017        $ENDIF
000018     $ENDIF
000019  $ENDIF
000020  $DEFAULT %1 TO '-#boot-intrinsic.lib'
000021  $DEFAULT %2 TO %1
000022  R{un}IUmanager
000023  %1
000024  %2
000025  I{nstall}%0
000026  QY
000027  $
000028  F{ile-MGR}B{ackup}%2,$
000029  Q{uit}
000030  $ENDEXEC
000031
000032

End of File -- Lines: 32 Characters: 744
```

```
================================================================================
FILE: "BUILD/MAKE/ATKLIB.TEXT"
================================================================================

000001  $EXEC {BUILD/MAKE/ATKLIB -- Assemble modules needed by the Toolkit}
000002  F{ile-Mgr}D{elete}LIBTK/XFER.OBJ
000003  Y{es}Q{uit}
000004  $SUBMIT BUILD/ASSEMB(LIBTK/XFER)
000005  $ENDEXEC
000006

End of File -- Lines: 6 Characters: 150
```

```
================================================================================
FILE: "BUILD/MAKE/CTK2LIB.TEXT"
================================================================================

000001  $EXEC {BUILD/MAKE/CTKLIB -- Compile units needed by the Toolkit}
000002  F{ile-Mgr}
000003  D{elete}LIBTK/UUNIVTEXT.OBJ
000004  Y{es}
000005  D{elete}LIBTK/UTEXT.OBJ
000006  Y{es}
000007  D{elete}LIBTK/UDIALOG.OBJ
000008  Y{es}
000009  Q{uit}
000010  $SUBMIT BUILD/COMP(LIBTK/UUNIVTEXT)
000011  $SUBMIT BUILD/COMP(LIBTK/UTEXT)
000012  $SUBMIT BUILD/COMP(LIBTK/UDIALOG)
000013  $ENDEXEC
000014

End of File -- Lines: 14 Characters: 277
```

```
================================================================================================
FILE: "BUILD/MAKE/CTKLIB.TEXT"
================================================================================================

000001  $EXEC {BUILD/MAKE/CTKLIB -- Compile units needed by the Toolkit}
000002  F{ile-Mgr}D{elete}LIBTK/UOBJECT.OBJ
000003  Y{es}
000004  D{elete}LIBTK/UDRAW.OBJ
000005  Y{es}
000006  D{elete}LIBTK/UABC.OBJ
000007  Y{es}
000008  Q{uit}
000009  $SUBMIT BUILD/COMP(LIBTK/UOBJECT)
000010  $SUBMIT BUILD/COMP(LIBTK/UDRAW)
000011  $SUBMIT BUILD/COMP(LIBTK/UABC)
000012  $ENDEXEC
000013

End of File -- Lines: 13 Characters: 267
```

```
================================================================================================
FILE: "BUILD/MAKE/LTK2LIB.TEXT"
================================================================================================

000001  $EXEC {BUILD/MAKE/LTKLIB -- Link the Toolkit}
000002  F{ile-Mgr}D{elete}-#boot-TK2LIB.OBJ
000003  Y{es}Q{uit}
000004  L{ink}?
000005  +i
000006  +m TKUTInit SgTxtIni
000007  +m DlgInit SgTxtIni
000008  +m DlgAlloc SgTxtIni
000009  +m SgTxtHot SgTxtRes
000010  +m TK2Start SgParRes
000011  +m SgTxtWrm SgParRes
000012  +m DlgText SgTxtRes
000013  +m SgTxtCld SgTxtTwo
000014  +m DlgDbg SgDIAdbg
000015  +m DlgHot SgDialog
000016  +m DlgRes SgDialog
000017  +m DlgCold SgDialog
000018  +m DlgWarm SgDialog
000019  +m HdgMarg SgDialog
000020  +m DlgLayou SgLayout
000021  +m TKUTWrit TKUT
000022  +m TKUTMain TKUT
000023
000024  LIBTK/UUNIVTEXT
000025  LIBTK/UTEXT
000026  LIBTK/UDIALOG
000027  -#boot-TKLIB
000028  -#boot-IOSPASLIB
000029  -#boot-SYS1LIB
000030  {no more input files}
000031  {no listing file}
000032  -#boot-TK2LIB
000033  $SUBMIT BUILD/INSTALL(11)
000034  $ENDEXEC
000035

End of File -- Lines: 35 Characters: 586
```

```
================================================================================================
FILE: "BUILD/MAKE/LTKLIB.TEXT"
================================================================================================

000001  $EXEC {BUILD/MAKE/LTKLIB -- Link the Toolkit}
000002  F{ile-Mgr}D{elete}-#boot-TKLIB.OBJ
000003  Y{es}Q{uit}
000004  L{ink}?
000005  +i
000006  +M SgCLAres SgABCdat
000007  +M SgCLAcld SgABCdat
000008  +M sABCdat SgABCdat {remove}
000009  +M sSplit SgABCdat
000010  +M sRes SgDRWres
000011  +M sClick SgDRWres {SgABCdat}
000012  +M sFilter SgDRWres {SgABCdat}
000013  +M SgXFER SgABCres
000014  +M sHotUtil SgABCres {?}
000015  +M sStartup SgABCres
000016  +M sResDat SgABCres
000017  +M sCommand SgDRWres {SgABCdat}
000018  +M sCmd2 SgABCres
000019  +M sScroll SgABCres
000020  +M sLOX SgCLAini
000021  +M sError SgABCdbg
000022  +M Override SgCLAini
000023  +M sCldInit SgCLAdbg
000024  +M sInit1 SgABCini
000025  +M sAlert SgABCcld
000026  +M sUtil SgCLAdbg
000027  +M sCut SgABCdat
000028  +M sPaste SgABCdat
000029
000030  LIBTK/UOBJECT
000031  LIBTK/UDRAW
000032  LIBTK/UABC
000033  LIBTK/XFER
000034  -#BOOT-IOSPASLIB
000035  -#BOOT-IOSFPLIB
000036  -#BOOT-SYS1LIB
000037  -#BOOT-PRLIB
000038  {no more input files}
000039  {no listing file}
000040  -#BOOT-TKLIB
000041  $SUBMIT BUILD/INSTALL(10)
000042  $ENDEXEC
000043
```

End of File -- Lines: 43 Characters: 753

```
================================================================================
FILE: "BUILD/MAKE/TKLIB.TEXT"
================================================================================

000001  $EXEC {BUILD/MAKE/TKLIB -- build the Toolkit}
000002  $SUBMIT BUILD/ASSEMB(LIBPL/CLASLIB)
000003  $SUBMIT BUILD/COMP(LIBPL/UCLASCAL)
000004  $SUBMIT BUILD/MAKE/ATKLIB
000005  $SUBMIT BUILD/MAKE/CTKLIB
000006  $SUBMIT BUILD/MAKE/LTKLIB
000007  $SUBMIT BUILD/MAKE/CTK2LIB
000008  $SUBMIT BUILD/MAKE/LTK2LIB
000009  $ENDEXEC

End of File -- Lines: 9 Characters: 249
```

```
================================================================================================
FILE: "INTERFACE/PASLIBCALL.TEXT"
================================================================================================

000001
000002  { libpl/paslibcall interface }
000003
000004      intrinsic;
000005
000006      interface
000007        USES
000008          {$U libos/syscall.obj }  syscall;
000009
000010        { ------------------------------------------------------------------------------}
000011
000012        CONST
000013          CclearScreen     = 1;                    {clear the whole screen}
000014          CclearEScreen    = 2;                    {clear to the end of the screen}
000015          CclearELine      = 3;                    {clear to end of line}
000016
000017          CgoHome          = 11;                   {move cursor to home position}
000018          CleftArrow       = 12;                   {move cursor left one character position}
000019          CrightArrow      = 13;                   {move cursor right one character position}
000020          CupArrow         = 14;                   {move cursor up one line position}
000021          CdownArrow       = 15;                   {move cursor down one line position}
000022
000023        { ------------------------------------------------------------------------------}
000024
000025          function PAbortFlag : boolean;      {Apple-period entered or not}
000026
000027          PROCEDURE GetGPrefix (Var prefix : pathname); {get global working directory}
000028
000029          procedure ScreenCtr (contrfun : integer); {standard screen control functions}
000030
000031          procedure GetPrDevice (var PrDevice : e_name);
000032
000033          function PaslibVersion : integer;     {return PASLIB version}
000034
000035          PROCEDURE PTranLisaChar (toTranslate : boolean);  {to translate Lisa char when print}
000036
000037            { Optional Call To Initialize the Heap }
000038          procedure PLINITHEAP(var ernum,refnum: integer; size,delta: longint;
000039                               ldsn: integer; swapable: boolean);
000040
000041      implementation
000042
000043      { FINIS }
```

000044

End of File -- Lines: 44 Characters: 1480

```
================================================================================================
FILE: "INTERFACE/PASSWD.TEXT"
================================================================================================

000001
000002  { libtk/passwd interface }
000003
000004      INTRINSIC;
000005
000006      {Provides calls for Password Protection in the Lisa Toolkit}
000007
000008
000009      { Copyright 1983, 1984, Apple Computer Inc. }
000010
000011      INTERFACE
000012
000013        USES {$U -#BOOT-SYSCALL} syscall;
000014
000015        procedure MAKE_SECURE ( var ecode  : integer;
000016                                var path   : pathname;
000017                                var passwd : e_name );
000018
000019        procedure KILL_SECURE ( var ecode  : integer;
000020                                var path   : pathname;
000021                                var passwd : e_name );
000022
000023        procedure OPEN_SECURE ( var ecode  : integer;
000024                                var path   : pathname;
000025                                var refnum : integer;
000026                                    manip  : mset;
000027                                var passwd : e_name );
000028
000029        procedure RENAME_SECURE ( var ecode   : integer;
000030                                  var path    : pathname;
000031                                  var newName : e_name;
000032                                  var passwd  : e_name );
000033
000034        procedure VERIFY_PASSWORD ( var ecode  : integer;
000035                                    var path   : pathname;
000036                                    var passwd : e_name );
000037
000038        procedure CHANGE_PASSWORD ( var ecode     : integer;
000039                                    var path      : pathname;
000040                                    var oldPasswd : e_name;
000041                                    var newPasswd : e_name );
000042
000043      IMPLEMENTATION
```

```
000044
000045      { FINIS }
000046

End of File -- Lines: 46 Characters: 1416
```

```
================================================================================================
FILE: "INTERFACE/PPASLIBC.TEXT"
================================================================================================

000001
000002  { libpl/ppaslibc interface }
000003
000004      intrinsic; interface
000005        USES {$U libos/syscall.obj } syscall;
000006
000007        type
000008         consoledest = (alscreen, mainscreen, xsorocA, xsorocB, folder, spare1, spare2, spare3); { max 8 }
000009         dsProcCode  = (dsResProg, dsSoftPwbtn, dsPrintDev, dsSetGPrefix, dsEnbDisk, dsGetDiskEnbF);
000010         dsProcParam = record
000011                         case ProcCode : dsProcCode of
000012                           dsResProg    : (RProcessId : longint); {must be called before
000013                                                                    the process starts running.}
000014                           dsSoftPwbtn  : (SPButton : boolean);                  {result}
000015                           dsPrintDev   : (PrDevice : e_name);
000016                           dsSetGPrefix : (errnum : INTEGER; prefix : pathname); {result}
000017                           dsEnbDisk    : (toEnbDisk : boolean);
000018                           dsGetDiskEnbF : (diskEnbF : boolean);                 {result}
000019                         end;
000020
000021      { ------ Procedures called by Shell only, some by WorkShop Shell only ---------------- }
000022
000023          PROCEDURE BlockIOInit;      {entire blockio unit init, once per system}
000024
000025          procedure BlockIOdisinit;     {blockio unit clean up, called by shell only}
000026
000027          procedure lockPaslib (var errnum : integer); {lock PASLIB1 for Filer}
000028
000029          procedure lockPasiolib (var errnum : integer); {lock PASIOLIB for Filer }
000030
000031          procedure moveconsole (var errnum : integer; applconsole : consoledest);
000032
000033          PROCEDURE ExecReset (VAR errnum : INTEGER; VAR execfile : pathname;
000034                               stopexec : BOOLEAN);   {open/stop exec file}
000035
000036          FUNCTION ExecFlag : BOOLEAN;        {return TRUE if EXEC file is active}
000037
000038          PROCEDURE OutputRedirect (VAR errnum : INTEGER; VAR outfile : pathname;
000039                                    stopoutput : BOOLEAN); {open/stop output file}
000040
000041          FUNCTION OutputRFlag : BOOLEAN;     {return true if output is redirected}
000042
000043          procedure DSPaslibCall (VAR ProcParam : dsProcParam); {by Workshop Shell only}
```

```
000044
000045      implementation
000046
000047    { FINIS }

End of File -- Lines: 47 Characters: 2018
```

```
================================================================================================
FILE: "LIBPL/CLASLIB.TEXT"
================================================================================================

000001  ; UNIT CLASLIB; {Copyright 1984, Apple Computer, Inc.}
000002
000003  ; {changed 02/06/84 1530    %_Method must swap in caller}
000004  ; {changed 01/20/84 1530    IUJSR decoded corrected}
000005  ; {changed 01/18/84 0732    Fixed BEQ bug in %_CallMethod & renamed it %_MethodCall}
000006  ; {changed 01/09/84 2105    Separated from XFER so we can include it in PASLIB
000007  ;                               SgPASres: %_CallMethod, %_Super, %GoLisaBug;
000008  ;                               SgPASini: %_JmpTo, %_ExitCaller, %_ExitPoppingTo, %_GetA5,
000009  ;                                         %_NextMethod; %_InsStack
000010  ;                            Added an argument to %_ExitPoppingTo}
000011
000012  ;============================================================================================
000013  DEBUGF  .EQU  1                 ; 1 to include $D+ info, 0 to exclude it
000014  ;============================================================================================
000015
000016          .MACRO HEAD
000017            .IF DEBUGF
000018              LINK    A6,#0       ; These two instructions form a slow no-op
000019              MOVE.L  (SP)+,A6
000020            .ENDC
000021          .ENDM
000022
000023          .MACRO TAIL
000024            .IF DEBUGF
000025              UNLK    A6
000026              RTS
000027              .ASCII  %1
000028            .ENDC
000029          .ENDM
000030
000031
000032  ;============================================================================================
000033          .SEG    'SgPASres'
000034  ;============================================================================================
000035
000036          .PROC  %_GoLisabug
000037          HEAD
000038
000039  ; PROCEDURE %_GoLisabug;
000040
000041          TRAP    #0
000042          RTS
000043
```

```
000044          TAIL    '%_GOLISA'
000045
000046  ;=======================================================================================
000047
000048          .FUNC   %_GetA5
000049          HEAD
000050
000051  ;
000052  ; FUNCTION %_GetA5: LONGINT;      { returns register A5 }
000053  ;
000054  ; USES  A0
000055  ;
000056          MOVE.L  (SP)+,A0        ; GET RETURN ADDRESS
000057          MOVE.L  A5,(SP)         ; STORE A5 INTO RETURN SLOT
000058          JMP     (A0)            ; EASY, HUH?
000059
000060          TAIL    '%_GETA5 '
000061
000062  ;=======================================================================================
000063
000064          .PROC %_MethodCall
000065          HEAD
000066
000067  ;    PROCEDURE %_MethodCall;        ; 157 cycles or about 32 microseconds for a regular call
000068
000069  ;       uses A0,A1,D0,D1,D2
000070
000071          MOVE.L  (SP)+,A1        ;08     A1 := Return Address
000072          TST.B   (A1)            ;08     Swap in caller
000073
000074          MOVE    #0,D0;          ;04     D0 := Level Number (0-origin)
000075          MOVE.B  (A1)+,D0        ;08
000076          LSL.W   #2,D0           ;10     Change to a byte offset
000077
000078          MOVE    #0,D1;          ;04     D1 := Method Number (1-origin)
000079          MOVE.B  (A1)+,D1        ;08
000080          LSL.W   #2,D1           ;10     Change to a byte offset
000081
000082          MOVE.L  A1,-(SP)        ;13     Return Address := A1 (which has been incremented by 2)
000083
000084          MOVE.L  4(SP),A0        ;16     A0 := SELF
000085
000086  .IF DEBUGF
000087          MOVE.L  A0,D2           ;04     MOVEA didn't set condition codes
000088          BEQ     SELFNIL         ;08     Error if NIL (next line fails anyway, but we could give a better msg}
000089  .ENDC
000090
000091          MOVE.L  (A0),A0         ;12     A0 := master pointer of SELF
```

```
000092          MOVE.L  (A0),A0         ;12     A0 := slice table pointer of SELF's class
000093
000094          MOVE.L  $00(A0,D0.W),A0 ;18     A0 := method table pointer for the desired level
000095          MOVE.L  -4(A0,D1.W),A0  ;18     A0 := method address
000096
000097          JMP     (A0)            ;08     Jump to method
000098
000099  SELFNIL DIVS    #0,D0           ;       **Temporary** Error report
000100
000101          TAIL    '%_METHOD'
000102
000103  ;=======================================================================================
000104
000105          .PROC %_SUPER
000106          HEAD
000107
000108  ;    PROCEDURE %_Super;         ; 199 cycles or about 44 microseconds for SUPERSELF (chain dist = 1)
000109
000110  ;       uses A0,A1,D0,D1,D2
000111
000112          MOVE.L  (SP)+,A1        ;08     A1 := Return Address
000113
000114          MOVE    #0,D1           ;04     D1 := Method Number (1-origin)
000115          MOVE.B  1(SP),D1        ;12
000116          LSL.W   #2,D1           ;10     Change to a byte offset
000117
000118          MOVE    #0,D0           ;04     D0 := Level Number (0-origin)
000119          MOVE.B  (SP)+,D0        ;08     Increments SP by 2!!
000120          LSL.W   #2,D0           ;10     Change to a byte offset
000121
000122          MOVE.W  (SP)+,D2        ;08     Chain distance
000123          MOVE.L  (SP)+,A0        ;12     Slice table pointer of this class
000124
000125          MOVE.L  A1,-(SP)        ;13     Return Address := A1 (which has not been modified)
000126
000127          JMP     ENDSUPL         ;10
000128
000129  SUPLOOP MOVE.L -4(A0),A0        ;16     A0 := superclass slice table pointer
000130  ENDSUPL DBEQ    D2,SUPLOOP      ;10-14  Loop until chain distance has been traversed (or end of chain)
000131
000132          MOVE.L  $00(A0,D0.W),A0 ;18     A0 := method table pointer for the desired level
000133          MOVE.L  -4(A0,D1.W),A0  ;18     A0 := method address
000134
000135          JMP     (A0)            ;08     Jump to method
000136
000137  SELFNIL DIVS #0,D0              ;       **Temporary** Error report
000138
000139          TAIL    '%_SUPER '
```

```
000140
000141
000142  ;=============================================================================================
000143          .SEG    'SgPASini'
000144  ;=============================================================================================
000145
000146
000147          .PROC %_JMPTO
000148          HEAD
000149
000150  ;    PROCEDURE %_JmpTo(pc: LONGINT);
000151
000152  ;       uses A0
000153
000154          MOVE.L  (SP)+,A0        ; Pop Return address and ignore it
000155          MOVE.L  (SP)+,A0        ; Pop pc argument
000156          JMP (A0)                ; Jump there
000157
000158          TAIL    '%_JMPTO '
000159
000160  ;=============================================================================================
000161
000162          .PROC %_EXITCA
000163          HEAD
000164
000165  ;    PROCEDURE %_ExitCaller;    that is, exit the caller of my caller, undoing two LINKs
000166
000167  ;       modifies A6,SP
000168
000169          UNLK    A6
000170          UNLK    A6
000171          RTS
000172
000173          .IF DEBUGF
000174          .ASCII  '%_EXITCA'
000175          .ENDC
000176
000177  ;=============================================================================================
000178
000179          .PROC %_EXITPO
000180          HEAD
000181
000182  ;   PROCEDURE %_ExitPoppingTo(newSP: LONGINT);
000183  ;                   exit my caller, and cut back the stack of the next frame to newSP
000184
000185  ;       uses A0,A1 and modifies A6,SP
000186
000187          MOVE.L  4(A6),A0        ; A0 := caller's return address
```

```
000188          MOVE.L  4(SP),A1        ; A1 := newSP
000189          UNLK    A6              ; pop my caller's stack frame
000190          MOVE.L  A1,SP           ; SP := newSP
000191          JMP     (A0)
000192
000193          .IF DEBUGF
000194          .ASCII  '%_EXITPO'
000195          .ENDC
000196
000197  ;=======================================================================================
000198
000199          .FUNC %_NextMethod
000200          HEAD
000201
000202  ;   FUNCTION  %_NextMethod(VAR pc@12: LONGINT;
000203  ;                          VAR impLevelNumber@8, impMethNumber@4: INTEGER
000204  ;                          )@16: ProcPtr;
000205
000206  ;       uses A0,A1,D0
000207
000208          MOVE.L  12(SP),A0       ; @PC
000209          MOVE.L  (A0),A1         ; PC throughout this routine
000210          TST.B   (A1)            ; swap in the code to test
000211
000212  INTRPLP CMP.W   #$4EBA,(A1)     ; test for JSR PC+d
000213          BEQ     JSR_PC
000214          CMP.W   #$4EAD,(A1)     ; test for JSR d(A5)
000215          BEQ     JSR_A5
000216          CMP.B   #$A0,(A1)       ; test for IUJSR
000217          BEQ     INTJSR
000218          CMP.W   #$3F3C,(A1)     ; test for MOVE.W #nn,-(SP)
000219          BEQ     PSHCON
000220          DIVS    #0,D0           ; supposedly impossible
000221
000222  PSHCON  MOVE.W  #0,D0           ; Clear D0 before loading a byte into it
000223          MOVE.B  2(A1),D0        ; D0 := the "Hi" of JSR PC+HiLo, i.e., levelNumber
000224          MOVE.L  8(SP),A0        ; A0 := @levelNumber
000225          MOVE.W  D0,(A0)         ; store levelNumber from D0
000226
000227          MOVE.W  #0,D0           ; Clear D0 before loading a byte into it
000228          MOVE.B  3(A1),D0        ; D0 := the "Lo" of JSR PC+HiLo, i.e., methodNumber
000229          SUB.W   #1,D0           ; decrement methodNumber (will be re-incremented by FINJSR)
000230          MOVE.L  4(SP),A0        ; A0 := @methodNumber
000231          MOVE.W  D0,(A0)         ; store methodNumber-1 from D0
000232
000233          ADD.L   #4,A1           ; increment PC past MOVE
000234          JMP     INTRPLP
000235
```

```
000236  INTJSR  MOVE.L  (A1),D1         ; D1 := IUJSR xxx
000237          AND.L   #$FFFFFF,D1     ; D1 := targetLocation
000238          MOVE.L  D1,A0           ; A0 := targetLocation
000239
000240  FINJSR  MOVE.L  A0,16(SP)       ; function result := targetLocation
000241
000242          ADD.L   #4,A1           ; increment PC past JSR
000243          MOVE.L  12(SP),A0       ; @PC
000244          MOVE.L  A1,(A0)         ; store back incremented PC
000245
000246          MOVE.L  4(SP),A0        ; A0 := @methodNumber
000247          ADD.W   #1,(A0)         ; increment methodNumber
000248
000249          MOVE.L  (SP)+,A0        ; pop and save return address
000250          ADD.L   #12,SP          ; pop and discard arguments
000251          JMP     (A0)            ; return
000252
000253  JSR_PC  MOVE.W  2(A1),D0        ; D0 := the "d" of JSR PC+d
000254          LEA     2(A1,D0.W),A0   ; A0 := targetLocation
000255          JMP     FINJSR
000256
000257  JSR_A5  MOVE.W  2(A1),D0        ; D0 := the "d" of JSR d(A5)
000258          LEA     0(A5,D0.W),A0   ; A0 := targetLocation
000259          JMP     FINJSR
000260
000261          TAIL    '%_NEXTME'
000262
000263  ;=============================================================================================
000264
000265          .FUNC %_InsStack
000266          HEAD
000267
000268  ;   PROCEDURE %_InsStack(addrToInsertAt, bytesToInsert: LONGINT);
000269  ;
000270  ;       This routine must be used with extreme care.  It inserts space in the middle of the stack.
000271  ;       It adjusts A6, A7, and the static chain, but it can not adjust other pointers that may
000272  ;       exist into the moved area; that is the responsibility of the caller.
000273  ;       This assumes that at least one static link needs adjustment
000274
000275  ;       uses A0,A1,D0,D1,D2; modifies A6,A7 and static chain
000276
000277          MOVE.L  (SP)+,D2        ; D2 := Return address
000278          MOVE.L  (SP)+,D1        ; D1 := bytesToInsert: must be even and at least 4
000279          MOVE.L  (SP)+,D0        ; D0 := addrToInsertAt: must be even
000280
000281          SUB.L   SP,D0           ; D0 := how many bytes need to move
000282          SUB.W   #2,D0           ; D0.W := how many longs
000283          LSR.W   #2,D0           ;         ... need to move
```

```
000284
000285          MOVE.L  SP,A0           ; A0 := Old SP
000286          SUB.L   D1,SP           ; SP := ultimate SP
000287          MOVE.L  SP,A1           ; A1 := ultimate SP
000288
000289          TST.W   -1024(SP)       ; Make the OS expand the stack if necessary
000290
000291  INSLP   MOVE.L  (A0)+,(A1)+     ; Move the data
000292          DBF     D0,INSLP
000293
000294          SUB.L   D1,A6           ; A6 := ultimate A6
000295          MOVE.L  A6,A1           ; A1 := addr of first static link
000296
000297  ADJLP   SUB.L   D1,(A1)         ; adjust this static link
000298          MOVE.L  (A1),A1         ; A1 := addr of next static link
000299          MOVE.L  (A1),D0         ; D0 := value of that static link
000300          CMP.L   A0,D0           ; If (value of that static link - first unmoved addr)
000301          BLT     ADJLP           ;       < 0 then that static link needs adjusting, too
000302
000303          MOVE.L  D2,A1           ; A1 := Return address
000304          JMP     (A1)            ; Return and Pray
000305
000306          TAIL    '%_INSSTA'
000307
000308  ;=======================================================================================
000309
000310          .END
000311
```

End of File -- Lines: 311 Characters: 10279

```
================================================================================================
FILE: "LIBPL/UCLASCAL.TEXT"
================================================================================================

000001  {UClascal -- In Spring '84 Release, part of PASLIB: only special units like UOBJECT will ever USE it}
000002  {Copyright 1984, Apple Computer, Inc.}
000003
000004  {changed 04/02/84 1330  Before exiting %_PGM2, see if the compiler saved A7 away, and if so
000005                              change the value to account for the method tables on the stack.}
000006  {changed 02/23/84 1200  %_InObCp/Cn: Make them work before classesInitialized, too}
000007  {changed 02/22/84 1715  CiToCn: Make it work before classesInitialized, too}
000008  {changed 02/19/84 1908  SizeOfCp & CiOfCp: Make them work before classesInitialized, too}
000009  {changed 01/18/84 2348  LookupInHashArray returns -index instead of -1 for failure, 0 for full table}
000010  {changed 01/18/84 0737  Renamed %_CallMethod to %_MethodCall so LisaBug traces mean more to people}
000011  {changed 01/15/84 1725  ObjectSize is always positive now, so QClassSize has been eliminated;
000012                          TOctet & TPOctets to INTERFACE}
000013  {changed 01/12/84 1952  Added TOctets and used it to get around signed byte bugs}
000014  {changed 01/12/84 1525  Added fTrcClascal to turn off extra writeLns when not debugging this unit;
000015                          %_InObCp/Cn mask off high byte of object's stp before testing quality}
000016  {changed 01/11/84 1714  Fixed a bunch of bugs}
000017  {changed 01/11/84 1312  Added DumpArrays}
000018  {changed 01/10/84 2117  More moved to UObject so apps don't have to USE this unit}
000019  {changed 01/05/84 2141  Began Construction}
000020
000021
000022  {   RESPONSIBILITIES...
000023
000024          The first class-init block is responsible for calling our procedure:
000025              InitClascal(PROCEDURE Finished(error: INTEGER));
000026          If no other class has already called it, then pleaseInitClascal will be TRUE, in case interested.
000027
000028          If an error occurs during initialization and InitClascal has been called, we'll call:
000029              Finished(error);
000030          The error code is an OS error code, except 3333 (need a new number!!!!) is our own error;
000031          If an error occurs during initialization and InitClascal has not been called, we
000032              do a Trap 0, which should get into LisaBug if present, else cause Technical Difficulties.
000033
000034          Just before returning from %_Pgm2, if FinishedProc is installed, we'll call:
000035              Finished(0);
000036          which may want to copy tables from pClasses, pSTables, pAuthors, pAliases (interface globals)
000037
000038          Typecast errors will also call Finished(3333) (need a new number!!!!)
000039
000040  }
000041
000042  {$SETC ForOS := TRUE }
000043
```

```
000044  UNIT UClascal;
000045
000046  {$SETC IsIntrinsic := TRUE }
000047
000048  {$IFC IsIntrinsic}
000049  INTRINSIC;
000050  {$ENDC}
000051
000052  INTERFACE
000053
000054  USES
000055      {$U -#BOOT-SysCall   } SysCall;
000056
000057
000058  {$SETC fTrcClascal   := FALSE}
000059  {$SETC fSymClascal   := TRUE}{FALSE}
000060  {$SETC fDbgClascal   := TRUE}{FALSE}
000061
000062
000063  {$%+}
000064
000065
000066  CONST
000067
000068      maxClasses  = 800;  {Hash table sizes}
000069      maxUnits    = 100;
000070
000071      maxAuthors  = 127;  {Because their indices are encoded in one byte in TClassInfo}
000072      maxAliases  = 127;
000073
000074
000075  TYPE
000076
000077      TByte = -128..127; {The T-names are so programs can USE UObject, NOT USE UClascal, and use "Byte"}
000078      TOctet = 0..255;
000079
000080      TOctets = PACKED ARRAY [0..32700] OF TOctet;
000081      TPOctets = ^TOctets;
000082
000083      TS8 = STRING[8];
000084      TS32 = STRING[32];
000085
000086      TA8 = PACKED ARRAY [1..8] OF CHAR;
000087      TA32 = PACKED ARRAY [1..32] OF CHAR;
000088
000089      THashCompare = (cHole, cMatch, cMismatch);
000090
000091      TMethodArray = ARRAY [1..256] OF LONGINT;
```

```
000092      TPMethodArray = ^TMethodArray;
000093
000094      TSliceTable = ARRAY [0..255] OF TPMethodArray;
000095      TPSliceTable = ^TSliceTable;
000096
000097      TClassInfo = RECORD             {16 bytes per class}
000098          classAlpha:             TA8;          {Class name in this program: Exactly 8 upper-case characters}
000099          superIndex:             INTEGER;      {Index of my superclass in ARRAY [1..xx] OF TClassInfo}
000100          objectSize:             INTEGER;      {SIZEOF(an object of this class) as declared}
000101          classAlias:             TByte;        {For ToolKit: Array index, or 0 if same as classAlpha}
000102          companyAndAuthor:       TByte;        {For ToolKit: Array index, or 0 if unspecified}
000103          version:                TByte;        {For ToolKit: Version number of the object format (default=1)}
000104          oldestReadableVersion:  TByte;        {For ToolKit: Oldest version number it is capable of updating}
000105          END;
000106
000107    {Each of the following types has only one member at run-time, and only during initialization}
000108    {These arrays start out small, but can grow.  Each has a single pointer that is updated automatically}
000109
000110      TClassArray     = ARRAY [1..maxClasses] OF TClassInfo;
000111      TPClassArray    = ^TClassArray;
000112
000113      TSTableArray    = ARRAY [1..maxClasses] OF TPSliceTable;
000114      TPSTableArray   = ^TSTableArray;
000115
000116      TAuthorArray    = ARRAY [1..maxAuthors] OF TA32;
000117      TPAuthorArray   = ^TAuthorArray;
000118
000119      TAliasArray     = ARRAY [1..maxAliases] OF TA8;
000120      TPAliasArray    = ^TAliasArray;
000121
000122
000123  VAR
000124
000125      pleaseInitClascal:  BOOLEAN;        {does InitClascal need to be called by some SUBCLASS OF NIL?}
000126      classesInitialized: BOOLEAN;        {has %_Pgm2 completed?}
000127
000128      pClasses:           TPClassArray;   {pointer to array of TClassInfo,   or NIL after %_Pgm2}
000129      pSTables:           TPSTableArray;  {.................of TPSliceTable, ..........}
000130      pAuthors:           TPAuthorArray;  {.................of TA32,         ..........}
000131      pAliases:           TPAliasArray;   {.................of TA8,          ..........}
000132      pMethods:           TPMethodArray;  {.................of ProcPtr,      ..........}
000133
000134      limClasses:         INTEGER;        {space allocated in pClasses^ & pSTables^}
000135      limAuthors:         INTEGER;        {................in pAuthors^}
000136      limAliases:         INTEGER;        {................in pAliases^}
000137      limMethods:         INTEGER;        {................in pMethods^}
000138
000139      numClasses:         INTEGER;        {number of elements in pClasses^ & pSTables^}
```

```
000140      numAuthors:         INTEGER;        {..................in pAuthors^}
000141      numAliases:         INTEGER;        {..................in pAliases^}
000142      numMethods:         INTEGER;        {..................in pMethods^   ... now or last time they existed}
000143
000144
000145  {Called from class-initialization blocks}
000146
000147  PROCEDURE InitClascal(PROCEDURE FinishedProc(error: INTEGER));          {required from first class-init}
000148
000149  PROCEDURE QUnitAuthor(VAR companyAndAuthor: TA32);                      {required once per unit with ToolKit}
000150  PROCEDURE QClassAuthor(VAR companyAndAuthor: TA32);                     {optional}
000151  PROCEDURE QClassAlias(VAR classAlias: TA8);                             {optional}
000152  PROCEDURE QClassVersion(itsVersion, oldestItCanRead: TByte);            {optional}
000153
000154
000155  {Called from version-conversion, allocation, and debugging code}
000156
000157  PROCEDURE CiToCn(index: INTEGER; VAR className: TS8);                   {convert class index to class title S8}
000158  PROCEDURE CpToCn(stp: TPSliceTable; VAR className: TS8);                {convert stp to class title S8}
000159
000160  FUNCTION  CiOfCp(stp: TPSliceTable): INTEGER;                           {convert stp to class index}
000161  FUNCTION  SizeOfCp(stp: TPSliceTable): INTEGER;                         {convert stp to object size}
000162
000163  FUNCTION  LookupInHashArray(tblSize: INTEGER; hashKey: LONGINT; toInsert: BOOLEAN;
000164                              FUNCTION Compare(index: INTEGER): THashCompare): INTEGER;
000165  FUNCTION  CallPC: LONGINT;
000166
000167
000168  {Called by code generated by the compiler}
000169
000170  PROCEDURE %_Pgm1;                       {Called before the first unit is initialized}
000171  PROCEDURE %_Unit;                       {Called at the beginning of each unit-initialization block}
000172  PROCEDURE %_Class(itsClassName, itsSuperName: TS8; itsSTP: TPSliceTable; itsEvenMethods, itsOddMethods,
000173                    itsObjSize: INTEGER); {Called at the beginning of each class-initialization block}
000174  PROCEDURE %_Pgm2;                       {Called after the last unit is initialized}
000175
000176  {These both return their first argument if it is NIL or passes a class-membership check; else Finished(3333)  }
000177  FUNCTION  %_CkObCP(ordObject, ordSTP: LONGINT): LONGINT;            {TFoo(obj),          TFoo in same unit }
000178  FUNCTION  %_CkObCN(ordObject: LONGINT; VAR className: TS8): LONGINT; {TFoo(obj),          TFoo in other unit}
000179
000180  {These both return TRUE if their first arg is NON-NIL and if it passes a class-membership check}
000181  FUNCTION  %_InObCP(ordObject, ordSTP: LONGINT): BOOLEAN;            {InClass(obj, TFoo), TFoo in same unit }
000182  FUNCTION  %_InObCN(ordObject: LONGINT; VAR className: TS8): BOOLEAN; {InClass(obj, TFoo), TFoo in other unit}
000183
000184
000185  IMPLEMENTATION
000186
000187
```

```
000188  {Segments: SgPASini(tialize and Terminate), SgPASres(ident)}
000189
000190  {$R-}
000191
000192  {$IFC fSymClascal}
000193  {$D+}
000194  {$ELSEC}
000195  {$D-}
000196  {$ENDC}
000197
000198
000199  CONST
000200
000201      minClasses  = 1;                {Initial array sizes}
000202      minAuthors  = 1;
000203      minAliases  = 1;
000204      minMethods  = 1;
000205
000206      growClasses = 100;              {Array growth increments (tunable)}
000207      growAuthors = 8;
000208      growAliases = 32;
000209      growMethods = 2000;
000210
000211
000212  TYPE
000213
000214      TS255 = STRING[255];
000215
000216      TPA8 = ^TA8;
000217      TPA32 = ^TA32;
000218
000219      TBytes = ARRAY [0..32700] OF TByte;
000220      TPBytes = ^TBytes;
000221
000222      TWords = ARRAY [0..16350] OF INTEGER;
000223      TPWords = ^TWords;
000224
000225      TPOctet = ^TOctet;
000226      TPByte = ^TByte;
000227      TPInt  = ^INTEGER;
000228      TPLint = ^LONGINT;
000229      Handle = ^TPLint;
000230
000231      ProcPtr = ^LONGINT;
000232
000233      TIdxArray = ARRAY [0..maxClasses] OF INTEGER;       {Element 0 is length; holes contain 0}
000234      TPIdxArray = ^TIdxArray;
000235
```

```
000236      TUnitArray = ARRAY [0..maxUnits] OF LONGINT;        {Element 0 is length; holes contain 0}
000237      TPUnitArray = ^TUnitArray;
000238
000239
000240  VAR
000241
000242      biggestAbstractClass:   INTEGER;        {max number of methods among all entirely abstract slices, or 1}
000243      mAllocAddr:             LONGINT;        {last allocated location in the method table}
000244      currCallCallPC:         LONGINT;        {the callPC of the unit whose classes are being initted}
000245
000246      pHashName:              TPIdxArray;     {index of a class in pClasses & pSTables, or 0 for a hole}
000247      pHashUnit:              TPUnitArray;    {pc of %_Unit caller, or 0 if a hole}
000248
000249      p%_Class:               ProcPtr;        {@ %_Class -- a pointer to the first instruction}
000250      pJmp%_Class:            ProcPtr;        {...same, but a pointer to the jump table entry}
000251
000252      pFinishedProc:          ProcPtr;        {@ FinishedProc passed in to InitClascal or @ DefaultFinishedProc}
000253      authorOfUnit:           TByte;          {Set by UnitAuthor and cleared by EndPreviousUnit}
000254      oldNumClasses:          INTEGER;        {numClasses at the beginning of this unit's initialization}
000255
000256      firstPackedName:        INTEGER;        {Set in %_Pgm2; see FindCn for explanation}
000257      dictBase:               LONGINT;        {Ditto}
000258
000259    {The following are assembler routines in CLASLIB.TEXT}
000260      FUNCTION  %_GetA5: LONGINT; EXTERNAL;
000261      FUNCTION  %_NextMethod(VAR pc: LONGINT;                                     {input and inc'd by 4 or 8}
000262                             VAR impLevelNumber, impMethNumber: INTEGER        {input and output both}
000263                             ): ProcPtr; EXTERNAL;
000264      PROCEDURE %_JmpTo(pc: LONGINT); EXTERNAL;
000265      PROCEDURE %_ExitCaller(argBytes: INTEGER); EXTERNAL;
000266      PROCEDURE %_ExitPoppingTo(newSP: LONGINT); EXTERNAL;
000267      PROCEDURE %_MethodCall; EXTERNAL;
000268      PROCEDURE %_GoLisaBug; EXTERNAL;
000269      PROCEDURE %_InsStack(addrToInsertAt, bytesToInsert: LONGINT); EXTERNAL;
000270
000271
000272  {$S SgPASres}
000273
000274
000275  FUNCTION  FindCn(index: INTEGER; VAR charsApart: BOOLEAN): LONGINT;
000276          {The class names starting with index=firstPackedName are stored packed below the method table,
000277              so the 8-character names start 8 bytes apart.
000278           The class names before index=firstSpreadName are stored in the unused high-order
000279              bytes of the method table, one every fourth byte, so the 8-character names start
000280              32 bytes apart.  This kludge saves 1-2K of resident storage in a ToolKit application.
000281           The name of the first class ends just before dictBase, the second class precedes it, etc.}
000282      VAR firstCharOffset: LONGINT;
000283  BEGIN
```

```
000284      charsApart := index < firstPackedName;
000285      IF charsApart THEN
000286          firstCharOffset := index * 32
000287      ELSE
000288          firstCharOffset := (numMethods * 4) + ((index + 1 - firstPackedName) * 8);
000289      FindCn := dictBase - firstCharOffset;
000290  END;
000291
000292
000293  FUNCTION  CiOfCp(stp: TPSliceTable): INTEGER;
000294      VAR index: INTEGER;
000295        {After init, the class index is recorded in the slice table, bytes 0 and 4 (high and low order bytes)}
000296  BEGIN
000297      CiOfCp := 0;
000298      IF classesInitialized THEN
000299          CiOfCp := 256 * TPOctets(stp)^[0] + TPOctets(stp)^[4]
000300      ELSE
000301          FOR index := 1 TO numClasses DO
000302              IF pSTables^[index] = stp THEN
000303                  CiOfCp := index;
000304  END;
000305
000306
000307  FUNCTION  SizeOfCp(stp: TPSliceTable): INTEGER;
000308        {After init, the size is recorded in the slice table, bytes 8 and 12, unless there are only 2 slices,}
000309        {..in which case the long before the slice table has a -1 in the even word and the object size
000310           in the odd word, instead of a superlink}
000311  BEGIN
000312      IF NOT classesInitialized THEN
000313          SizeOfCp := pClasses^[CiOfCp(stp)].objectSize
000314      ELSE
000315      IF TPWords(stp)^[-2] <= 0 THEN
000316          SizeOfCp := TPWords(stp)^[-1]
000317      ELSE
000318          SizeOfCp := 256 * TPOctets(stp)^[8] + TPOctets(stp)^[12];
000319  END;
000320
000321
000322  PROCEDURE CiToCn(index: INTEGER; VAR className: TS8);
000323      VAR charsApart: BOOLEAN;
000324          deltaAddr:  INTEGER;
000325          dictAddr:   LONGINT;
000326          i:          INTEGER;
000327          classAlpha: TA8;
000328  BEGIN
000329      className[0] := CHAR(8);
000330      IF NOT classesInitialized THEN
000331          BEGIN
```

```
000332          classAlpha := pClasses^[index].classAlpha;
000333          FOR i := 1 TO 8 DO
000334              className[i] := classAlpha[i];
000335          END
000336      ELSE
000337          BEGIN
000338          dictAddr := FindCn(index, charsApart);
000339          deltaAddr := 3*ORD(charsApart) + 1;
000340          FOR i := 1 TO 8 DO
000341              BEGIN
000342              className[i] := CHAR(TPByte(dictAddr)^);
000343              dictAddr := dictAddr + deltaAddr;
000344              END;
000345          END;
000346  END;
000347
000348
000349  PROCEDURE CpToCn(stp: TPSliceTable; VAR className: TS8);
000350  BEGIN
000351      CiToCn(CiOfCp(stp), className);
000352  END;
000353
000354
000355  PROCEDURE DefaultFinishedProc(error: INTEGER);
000356  BEGIN
000357      %_GoLisabug;
000358  END;
000359
000360
000361  PROCEDURE CallFinishedProc(PROCEDURE ModelFinishedProc(error: INTEGER); error: INTEGER);
000362      VAR pModelFinishedProc: TPLint;
000363  BEGIN
000364      pModelFinishedProc  := TPLint(ORD(@pModelFinishedProc) + 18);
000365      pModelFinishedProc^ := ORD(pFinishedProc);
000366      ModelFinishedProc(error);
000367  END;
000368
000369
000370  PROCEDURE CLAFail(error: INTEGER);                              {Called when fDbgClascal is FALSE}
000371  BEGIN
000372      IF error = 0 THEN
000373          error := 3333;                                          {GET A NUMBER ASSIGNED}
000374      CallFinishedProc(DefaultFinishedProc, error);
000375  END;
000376
000377
000378  {$IFC fDbgClascal}
000379  PROCEDURE CLABreak(s: TS255; n: LONGINT);        {Called when fDbgClascal is TRUE}
```

```
000380  BEGIN
000381      WriteLn('CLABreak: ', s, ' = ', n);
000382      ClaFail(0);
000383  END;
000384  {$ENDC}
000385
000386
000387  {Each expression "InClass(obj, TFoo)" generates:
000388      %_InObCp(val, classPtr) or %_InObCn(val, 'TFOO    ')
000389      The former ("In Object Class Pointer") is generated when TFoo is defined in the same unit.
000390      The latter ("In Object Class Name") is generated when TFoo is defined in another unit.
000391      Both are defined below}
000392
000393  FUNCTION %_InObCp(ordObject, ordSTP: LONGINT): BOOLEAN;
000394      TYPE    PST = ^TST;
000395              TST = ARRAY[0..0] OF PST;
000396              PPST = ^PST;
000397              PPPST = ^PPST;
000398      VAR trialSTP:       PST;
000399          pSTP:           PPST;
000400  BEGIN
000401      %_InObCp := FALSE;
000402      IF ordObject <> 0 THEN
000403          BEGIN
000404          trialSTP := PPPST(ordObject)^^;
000405          pSTP := @trialSTP;
000406          TPByte(pSTP)^ := 0;
000407          WHILE trialSTP <> PST(ordSTP) DO
000408              BEGIN
000409              IF classesInitialized THEN
000410                  trialSTP := trialSTP^[-1]
000411              ELSE
000412                  trialSTP := PST (TPMethodArray(pSTables^[pClasses^[CiOfCp(TPSliceTable(trialSTP))].superIndex]));
000413              IF ORD(trialSTP) <= 0 THEN
000414                  EXIT(%_InObCp);
000415              END;
000416          %_InObCp := TRUE;
000417          END;
000418  END;
000419
000420
000421  FUNCTION %_InObCn(ordObject: LONGINT; VAR className: TS8): BOOLEAN;
000422      TYPE    PST = ^TST;
000423              TST = ARRAY[0..0] OF PST;
000424              PPST = ^PST;
000425              PPPST = ^PPST;
000426      VAR trialSTP:       PST;
000427          tryClassName:   TS8;
```

```
000428          pSTP:           PPST;
000429  BEGIN
000430      %_InObCn := FALSE;
000431      IF ordObject <> 0 THEN
000432          BEGIN
000433          trialSTP := PPPST(ordObject)^^;
000434          pSTP := @trialSTP;
000435          TPByte(pSTP)^ := 0;
000436          REPEAT
000437              CpToCn(TPSliceTable(trialSTP), tryClassName);
000438              IF tryClassName = className THEN
000439                  BEGIN
000440                  %_InObCn := TRUE;
000441                  EXIT(%_InObCn);
000442                  END;
000443              IF classesInitialized THEN
000444                  trialSTP := trialSTP^[-1]
000445              ELSE
000446                  trialSTP := PST(TPMethodArray(pSTables^[pClasses^[CiOfCp(TPSliceTable(trialSTP))].superIndex]));
000447            UNTIL ORD(trialSTP) <= 0;
000448          END;
000449  END;
000450
000451
000452  {Each typecast expression TFoo(val) with range checking on generates:
000453      %_CkObCp(val, classPtr) or %_CkObCn(val, 'TFOO    ')
000454      The former ("Check Object Class Pointer") is generated when TFoo is defined in the same unit.
000455      The latter ("Check Object Class Name") is generated when TFoo is defined in another unit.
000456      Both are defined below}
000457
000458  FUNCTION %_CkObCp(ordObject, ordSTP: LONGINT): LONGINT;
000459      VAR objClassName:   TS8;
000460          desClassName:   TS8;
000461  BEGIN
000462      %_CkObCp := ordObject;
000463      IF ordObject <> 0 THEN
000464          IF NOT %_InObCp(ordObject, ordSTP) THEN
000465              BEGIN
000466              CpToCn(TPSliceTable(Handle(ordObject)^^), objClassName);
000467              CpToCn(TPSliceTable(ordSTP), desClassName);
000468              {$IFC fDbgClascal}
000469              CLABreak(CONCAT('Attempt to coerce an object of class ',
000470                              CONCAT(objClassName,
000471                                     CONCAT(' to a value of type ',
000472                                            desClassName))),
000473                       0);
000474              {$ELSEC}
000475              CLAFail(0);
```

```
000476              {$ENDC}
000477              EXIT(%_CkObCp);
000478              END;
000479  END;
000480
000481
000482  FUNCTION %_CkObCn(ordObject: LONGINT; VAR className: TS8): LONGINT;
000483      VAR objClassName:   TS8;
000484  BEGIN
000485      %_CkObCn := ordObject;
000486      IF ordObject <> 0 THEN
000487          IF NOT %_InObCn(ordObject, className) THEN
000488              BEGIN
000489              CpToCn(TPSliceTable(Handle(ordObject)^^), objClassName);
000490              {$IFC fDbgClascal}
000491              CLABreak(CONCAT('Attempt to coerce an object of class ',
000492                              CONCAT(objClassName,
000493                                     CONCAT(' to a value of type ',
000494                                            className))),
000495                       0);
000496              {$ELSEC}
000497              CLAFail(0);
000498              {$ENDC}
000499              EXIT(%_CkObCn);
000500              END;
000501  END;
000502
000503
000504  {MUST BE IN A DIFFERENT SEGMENT FROM %_Class, i.e., NOT IN SgPASini}
000505  FUNCTION GetPJmp%_Class: ProcPtr;
000506  BEGIN
000507      GetPJmp%_Class := @%_Class;
000508  END;
000509
000510
000511  {$S SgPASini}
000512
000513
000514  PROCEDURE InitClascal(PROCEDURE FinishedProc(error: INTEGER));
000515  BEGIN
000516      pFinishedProc    := @FinishedProc;
000517      pleaseInitClascal := FALSE;
000518  END;
000519
000520
000521  PROCEDURE StoreCn(index: INTEGER; VAR classAlpha: TA8);
000522      VAR charsApart: BOOLEAN;
000523          dictAddr:   LONGINT;
```

```
000524          i:          INTEGER;
000525  BEGIN
000526      dictAddr := FindCn(index, charsApart);
000527      IF charsApart THEN
000528          FOR i := 1 TO 8 DO
000529              BEGIN
000530              TPByte(dictAddr)^ := TByte(classAlpha[i]);
000531              dictAddr := dictAddr + 4;
000532              END
000533      ELSE
000534          TPA8(dictAddr)^ := classAlpha;
000535  END;
000536
000537
000538  PROCEDURE _Abstract;
000539  BEGIN
000540      {$IFC fDbgClascal}
000541      CLABreak('An ABSTRACT method has been called: you can''t continue', 0);
000542      {$ELSEC}
000543      CLAFail(0);
000544      {$ENDC}
000545  END;
000546
000547
000548  PROCEDURE InsStack(addrOfGrownArray, afterByte, bytesToInsert: LONGINT);
000549
000550      PROCEDURE AdjustPArray(VAR addrOfOtherArray: LONGINT; which: TS32);
000551      BEGIN
000552          {$IFC fTrcClascal}
000553          Write('... ', which, ' moved from ', addrOfOtherArray:12, ' to ');
000554          {$ENDC}
000555          IF (addrOfGrownArray + afterByte) >= addrOfOtherArray THEN
000556              addrOfOtherArray := addrOfOtherArray - bytesToInsert;
000557          {$IFC fTrcClascal}
000558          WriteLn(addrOfOtherArray:12);
000559          {$ENDC}
000560      END;
000561
000562  BEGIN
000563      {$IFC fTrcClascal}
000564      WriteLn('$$$ About to insert ', bytesToInsert:4, ' bytes after byte ', afterByte:3,
000565              ' of ', addrOfGrownArray:5, '$$$');
000566      {$ENDC}
000567      %_InsStack(addrOfGrownArray + afterByte, bytesToInsert); {bytesToInsert must be even and at least 4}
000568      AdjustPArray(LONGINT(pAuthors), 'pAuthors');
000569      AdjustPArray(LONGINT(pAliases), 'pAliases');
000570      AdjustPArray(LONGINT(pClasses), 'pClasses');
000571      AdjustPArray(LONGINT(pSTables), 'pSTables');
```

```
000572      AdjustPArray(LONGINT(pMethods), 'pMethods');
000573      AdjustPArray(LONGINT(pHashName),'pHashName');
000574      AdjustPArray(LONGINT(pHashUnit),'pHashUnit');
000575  END;
000576
000577
000578  FUNCTION MAllocate(numNeeded, numToGrowBy: INTEGER): LONGINT;
000579          {** NO VAR PARAMETERS ALLOWED THAT ARE REFERENCED AFTER CALLING InsStack **}
000580      VAR numBytes:       LONGINT;
000581          bytesToInsert:  LONGINT;
000582  BEGIN
000583      numBytes := 4 * numNeeded;
000584      mAllocAddr := mAllocAddr - numBytes;
000585      MAllocate := mAllocAddr;
000586      bytesToInsert := ORD(pMethods) - mAllocAddr;
000587      IF bytesToInsert > 0 THEN
000588          BEGIN
000589          IF bytesToInsert < (4 * numToGrowBy) THEN
000590              bytesToInsert := 4 * numToGrowBy;
000591          InsStack(ORD(pMethods), 0, bytesToInsert);
000592          END;
000593      {$IFC fTrcClascal}
000594      WriteLn('******* Allocated ', numNeeded:3, ' method entries at ', mAllocAddr:5, '********');
000595      {$ENDC}
000596  END;
000597
000598
000599  FUNCTION RAllocate(bytesPerRec, numNow, numToGrowBy, numRoomFor, maxNumAllowed: INTEGER;
000600                     whutzits: TS8; ordPArray: LONGINT): INTEGER;
000601          {** NO VAR PARAMETERS ALLOWED THAT ARE REFERENCED AFTER CALLING InsStack **}
000602          {bytesPerRec must be even; this function returns the new numRoomFor value}
000603      VAR bytesToInsert: INTEGER;
000604  BEGIN
000605      IF (numRoomFor + numToGrowBy) > maxNumAllowed THEN
000606          numToGrowBy := maxNumAllowed - numRoomFor;
000607
000608      IF numToGrowBy <= 0 THEN
000609          {$IFC fDbgClascal}
000610          CLABreak(CONCAT('Too many ', whutzits), maxNumAllowed);
000611          {$ELSEC}
000612          CLAFail(0);
000613          {$ENDC}
000614
000615      bytesToInsert := bytesPerRec * numToGrowBy;
000616      InsStack(ordPArray, bytesPerRec * numNow, bytesToInsert);
000617      RAllocate := numRoomFor + numToGrowBy;
000618  END;
000619
```

```
000620
000621  FUNCTION LookupAuthor(VAR classAuthor: TA32): INTEGER;
000622          {There should be room for two Authors (a ClassAuthor & a UnitAuthor) because %_Class checked}
000623      VAR addr:   LONGINT;
000624          i:      INTEGER;
000625  BEGIN
000626      addr := ORD(pAuthors);
000627      FOR i := 1 TO numAuthors DO
000628          BEGIN
000629          IF TPA32(addr)^ = classAuthor THEN
000630              BEGIN
000631              LookupAuthor := i;
000632              EXIT(LookupAuthor);
000633              END;
000634          addr := addr + 32;
000635          END;
000636
000637      IF numAuthors >= limAuthors THEN
000638          CLAFail(0)
000639      ELSE
000640          BEGIN
000641          numAuthors := numAuthors + 1;
000642          TPA32(addr)^ := classAuthor;
000643          LookupAuthor := numAuthors;
000644          END;
000645  END;
000646
000647
000648  {** I tried merging the routines above and below, but I don't think it is worth it **}
000649
000650
000651  FUNCTION LookupAlias(VAR classAlias: TA8): INTEGER;
000652          {There should be room for one alias because %_Class checked}
000653      VAR addr:   LONGINT;
000654          i:      INTEGER;
000655  BEGIN
000656      addr := ORD(pAliases);
000657      FOR i := 1 TO numAliases DO
000658          BEGIN
000659          IF TPA8(addr)^ = classAlias THEN
000660              BEGIN
000661              LookupAlias := i;
000662              EXIT(LookupAlias);
000663              END;
000664          addr := addr + 8;
000665          END;
000666
000667      IF numAliases >= limAliases THEN
```

```
000668          CLAFail(0)
000669      ELSE
000670          BEGIN
000671          numAliases := numAliases + 1;
000672          TPA8(addr)^ := classAlias;
000673          LookupAlias := numAliases;
000674          END;
000675  END;
000676
000677
000678  PROCEDURE QUnitAuthor(VAR companyAndAuthor: TA32);
000679  BEGIN
000680      IF classesInitialized THEN
000681          CLAFail(0);
000682      authorOfUnit := LookupAuthor(companyAndAuthor);
000683  END;
000684
000685
000686  PROCEDURE QClassAuthor(VAR companyAndAuthor: TA32);
000687  BEGIN   {Must call procedures before the WITH because Lookups might move pClasses^}
000688      IF classesInitialized THEN
000689          CLAFail(0);
000690      pClasses^[numClasses].companyAndAuthor := LookupAuthor(companyAndAuthor);
000691  END;
000692
000693
000694  PROCEDURE QClassAlias(VAR classAlias: TA8);
000695  BEGIN   {Must call procedures before the WITH because Lookups might move pClasses^}
000696      IF classesInitialized THEN
000697          CLAFail(0);
000698      pClasses^[numClasses].classAlias := LookupAlias(classAlias);
000699  END;
000700
000701
000702  PROCEDURE QClassVersion(itsVersion, oldestItCanRead: TByte);
000703  BEGIN
000704      IF classesInitialized THEN
000705          CLAFail(0);
000706      WITH pClasses^[numClasses] DO
000707          BEGIN
000708          version := itsVersion;
000709          oldestReadableVersion := oldestItCanRead;
000710          END;
000711  END;
000712
000713
000714  FUNCTION NumSlices(classIndex: INTEGER): INTEGER;
000715      VAR n: INTEGER;
```

```
000716  BEGIN
000717      n := 0;
000718      WHILE classIndex > 0 DO
000719          BEGIN
000720          classIndex := pClasses^[classIndex].superIndex;
000721          n := n + 2;
000722          END;
000723      NumSlices := n;
000724  END;
000725
000726
000727  FUNCTION  CallCallPC: LONGINT;
000728      VAR dummy:      INTEGER;    { must be first local and two bytes long }
000729  BEGIN
000730      CallCallPC := TPLint(TPLint(TPLint(ORD(@dummy) + 2)^)^ + 4)^;   {caller's caller's return address}
000731  END;
000732
000733
000734  FUNCTION  CallPC: LONGINT;
000735      VAR dummy:      INTEGER;    { must be first local and two bytes long }
000736  BEGIN
000737      CallPC := TPLint(TPLint(ORD(@dummy) + 2)^ + 4)^;            {caller's return address}
000738  END;
000739
000740
000741  PROCEDURE SetCallPC(pc: LONGINT);
000742      VAR dummy:      INTEGER;    { must be first local and two bytes long }
000743          addrOfPC:   LONGINT;
000744  BEGIN
000745      addrOfPC := TPLint(ORD(@dummy) + 2)^ + 4;
000746      TPLint(addrOfPC)^ := pc;          {caller's return address}
000747  END;
000748
000749
000750  FUNCTION LookupInHashArray(tblSize: INTEGER; hashKey: LONGINT; toInsert: BOOLEAN;
000751                  FUNCTION Compare(index: INTEGER): THashCompare): INTEGER;
000752          {toInsert, return: -index if entry already there, index (>0) if a hole found}
000753          {not toInsert, return: index (> 0) if entry found, -index if not there}
000754          {return 0 if table is full}
000755      VAR probe:          INTEGER;
000756          origProbe:      INTEGER;
000757          hashCompare:    THashCompare;
000758  BEGIN               {This could be made faster -- and probably should be}
000759      LookupInHashArray := 0;
000760      probe := hashKey;
000761      probe := (ABS(probe) MOD tblSize) + 1;
000762      origProbe := probe;
000763      REPEAT
```

```
000764          hashCompare := Compare(probe);
000765          IF hashCompare <> cMismatch THEN
000766              BEGIN
000767              IF toInsert = (hashCompare = cHole) THEN
000768                  LookupInHashArray := probe
000769              ELSE
000770                  LookupInHashArray := - probe;
000771              EXIT(LookupInHashArray);
000772              END;
000773          probe := probe + 1;
000774          IF probe > tblSize THEN
000775              probe := 1;
000776        UNTIL probe = origProbe;
000777  END;
000778
000779
000780  {$IFC fTrcClascal}
000781  PROCEDURE DumpArrays;
000782      VAR index:          INTEGER;
000783          itsSTP:         TPSliceTable;
000784          slices:         INTEGER;
000785          s:              TS8;
000786          j:              INTEGER;
000787          i:              INTEGER;
000788          level:          INTEGER;
000789          methArrPtr:     TPMethodArray;
000790          numAtThatLevel: INTEGER;
000791  BEGIN
000792      WriteLn;
000793      WriteLn(' *************** ARRAYS *************** ');
000794      WriteLn;
000795      FOR index := 1 TO numClasses DO
000796          BEGIN
000797          Write('Class Index = ', index:3);
000798          itsSTP := pSTables^[index];
000799          Write('  Class Pointer = ', ORD(itsSTP):10);
000800          slices := NumSlices(index);
000801          Write('  Number of slices = ', slices:3);
000802          s[0] := CHAR(8);
000803          FOR j := 1 TO 8 DO
000804              s[j] := pClasses^[index].classAlpha[j];
000805          WriteLn('  Name = ', s);
000806          i := index;
000807          FOR level := slices - 1 DOWNTO 0 DO
000808              BEGIN
000809              Write('   Level ', level:1);
000810              Write('   Index ', i:2);
000811
```

```
000812              methArrPtr := itsSTP^[level];
000813              Write(' Method array ptr = ', ORD(methArrPtr):10);
000814
000815              numAtThatLevel := TPWords(pSTables^[i])^[ORD(ODD(level))-2];
000816              Write(' numAtThatLevel ', numAtThatLevel:2);
000817
000818              IF methArrPtr = NIL THEN
000819                  WriteLn(', ... all Abstract')
000820              ELSE
000821                  BEGIN
000822                  WriteLn;
000823                  FOR j := 1 TO numAtThatLevel DO
000824                      WriteLn(j:10, ORD(methArrPtr^[j]):10);
000825                  END;
000826
000827              IF NOT ODD(level) THEN
000828                  i := pClasses^[i].superIndex;
000829              WriteLn;
000830              END;
000831          WriteLn;
000832          END;
000833  END;
000834  {$ENDC}
000835
000836
000837  {The main program starts with:
000838      JSR %_Pgm1      ; Defined below
000839      JSR unit#m      ; for every unit USEd by the main program within $CLASSES+ (in order USEd)...
000840      ...
000841      JSR unit#n
000842      JSR %_Pgm2      ; Defined below}
000843
000844  PROCEDURE %_Pgm1;
000845      VAR methads:        ARRAY [1..minMethods] OF ProcPtr;     {!!! MUST MUST MUST be the first VAR !!!}
000846          aliases:        ARRAY [1..minAliases] OF TA8;         {!!! Should be in this group of VARs !!!}
000847          authors:        ARRAY [1..minAuthors] OF TA32;        {!!! Should be in this group of VARs !!!}
000848          sTables:        ARRAY [1..minClasses] OF TPSliceTable;{!!! Should be in this group of VARs !!!}
000849          classes:        ARRAY [1..minClasses] OF TClassInfo;  {!!! Should be in this group of VARs !!!}
000850        {The arrays above can grow; only one ptr to each is maintained in a global variable, e.g., pMethods}
000851          excepName:      T_Ex_Name;      {These all stay allocated until the end of %_Pgm2}
000852          error:          INTEGER;
000853          addr:           LONGINT;
000854          i:              INTEGER;
000855          hashUnit:       TUnitArray;
000856          hashName:       TIdxArray;
000857
000858  BEGIN
000859     {Install Default Finished procedure}
```

```
000860      pFinishedProc := @DefaultFinished;
000861
000862     {Initialize global interface variables}
000863
000864      pleaseInitClascal := TRUE;      {A global set to FALSE in InitClascal}
000865      classesInitialized := FALSE;    {A global set TRUE in %_Pgm2}
000866
000867      pClasses := @classes;
000868      pSTables := @sTables;
000869      pAuthors := @authors;
000870      pAliases := @aliases;
000871      pMethods := @methads;   {methads spelled funny because METHODS is a reserved word}
000872          {NOTE: pMethods^[] is never written; the "ARRAY" can be > 32K bytes if necessary}
000873
000874      limClasses := minClasses;
000875      limAuthors := minAuthors;
000876      limAliases := minAliases;
000877      limMethods := minMethods;
000878
000879      numClasses := 0; {incremented by %_Class}
000880      numAuthors := 0; {never modified in this unit; UOBJECT manages them}
000881      numAliases := 0; {never modified in this unit; UOBJECT manages them}
000882      numMethods := 0; {incremented by FillArraysFrom, called by %_Class}
000883
000884     {Set the scheduling mode}
000885      Sched_Class(error, TRUE);
000886      IF error > 0 THEN
000887          CLAFail(error);
000888
000889     {Set six bytes at 0(A5) to JMP %_MethodCall in XFER}
000890      addr := %_GetA5;
000891      TPInt(addr)^ := $4EF9; {JMP fullAddr}
000892      addr := addr + 2;
000893      TPLint(addr)^ := ORD(@%_MethodCall);
000894
000895     {Clear hash tables}
000896      FOR i := 1 TO maxUnits DO
000897          hashUnit[i] := 0;
000898      FOR i := 1 TO maxClasses DO
000899          hashName[i] := 0;
000900
000901     {Initialize global implementation variables}
000902
000903      pHashName  := @hashName;
000904      pHashUnit  := @hashUnit;
000905
000906      authorOfUnit := 0;
000907      mAllocAddr := ORD(pMethods) + limMethods * 4;
```

```
000908      biggestAbstractClass := 1;      {Could be 0, but this produces a more comprehensible memory dump}
000909      currCallCallPC := 0;
000910
000911      p%_Class    := @%_Class;        {The %_NextMethod loop in %_Class stops at a JSR %_Class}
000912      pJmp%_Class := GetPJmp%_Class;  {A function in another segment must get the jump table address for me}
000913
000914     {We can never return because we need our locals around during the unit initializations and need
000915         the method tables around forever}
000916      %_JmpTo(CallPC);
000917  END;
000918
000919
000920  PROCEDURE EndPreviousUnit;  {We don't require companyAndAuthor--but client could do so at the end of %_Pgm2}
000921      VAR i: INTEGER;
000922  BEGIN
000923      IF authorOfUnit <> 0 THEN
000924          FOR i := oldNumClasses + 1 TO numClasses DO
000925              WITH pClasses^[i] DO
000926                  IF companyAndAuthor = 0 THEN
000927                      companyAndAuthor := authorOfUnit;
000928      authorOfUnit := 0;
000929      oldNumClasses := numClasses;
000930  END;
000931
000932
000933  PROCEDURE %_Pgm2;
000934          {** NO VAR PARAMETERS ALLOWED THAT ARE REFERENCED AFTER CALLING MAllocate **}
000935      VAR dummy:          LONGINT; {MUST BE FIRST VAR AND 4 BYTES LONG!!!}
000936          pAbstracts:     TPMethodArray;
000937          index:          INTEGER;
000938          extraLongs:     LONGINT;
000939          itsSTP:         TPSliceTable;
000940          slices:         INTEGER;
000941          level:          INTEGER;
000942          objSize:        INTEGER;
000943          pInt:           TPInt;
000944          pLint:          TPLint;
000945  BEGIN
000946      EndPreviousUnit;
000947
000948    {For any slice that was fully abstract, we will make it point at a special block of @_Abstract}
000949      pAbstracts := TPMethodArray(MAllocate(biggestAbstractClass, 16));
000950      numMethods := numMethods + biggestAbstractClass;
000951      FOR index := 1 TO biggestAbstractClass DO
000952          pAbstracts^[index] := ORD(@ _Abstract);
000953
000954    {Assure sufficient room for names}
000955      dictBase := mAllocAddr + (numMethods * 4);
```

```
000956      firstPackedName := (numMethods DIV 8) + 1;
000957      extraLongs := 2 * (numClasses - firstPackedName + 1);
000958      IF extraLongs > 0 THEN
000959          dummy := MAllocate(extraLongs, 0);
000960
000961      {$IFC fTrcClascal}
000962      WriteLn('biggestAbstractClass   = ', biggestAbstractClass:6);
000963      WriteLn('numMethods allocated   = ', numMethods:6);
000964      WriteLn('firstPackedName        = ', firstPackedName:6);
000965      WriteLn('extraLongs             = ', extraLongs:6);
000966      WriteLn('mAllocAddr             = ', mAllocAddr:6);
000967      WriteLn('dictBase               = ', dictBase:6);
000968      WriteLn('pClasses               = ', ORD(pClasses):6);
000969      WriteLn('pSTables               = ', ORD(pSTables):6);
000970      {$ENDC}
000971
000972    {Search back from call to %_PGM2 for a MOVE.L A7, xxxx(A5) (opcode $2B4F); if found, calculate the
000973         address that contains the saved A7 and stuff in mAllocAddr instead.  Stop searching if we
000974         find a LINK A5, xxxx instruction.}
000975      pLint := Pointer(Ord(@dummy) + 8); {pLint^ should be our return address}
000976      pInt := Pointer(pLint^);
000977
000978      WHILE (pInt^ <> $2B4F {MOVE.L A7, xxxx(A5)}) AND (pInt^ <> $4E55 {LINK A5, xxxx}) DO
000979          pInt := Pointer(Ord(pInt) - 2);
000980      IF pInt^ = $2B4F THEN
000981          BEGIN
000982          pInt := Pointer(Ord(pInt) + 2);
000983          pLint := Pointer(pInt^ + %_GetA5);
000984          pLint^ := mAllocAddr;
000985          END;
000986
000987    {Final initialization of each class in turn}
000988      FOR index := 1 TO numClasses DO
000989          BEGIN
000990        {Fill in missing slices}
000991          itsSTP := pSTables^[index];
000992          slices := NumSlices(index);
000993          FOR level := 0 TO slices - 1 DO
000994              IF itsSTP^[level] = NIL THEN
000995                  itsSTP^[level] := pAbstracts;
000996
000997        {Copy the name to the method table area}
000998          StoreCn(index, pClasses^[index].classAlpha);
000999
001000        {The class index is recorded in the slice table, bytes 0 and 4 (high and low order bytes)}
001001        {The object size is recorded in the slice table, bytes 8 and 12, unless there are only two slices,}
001002        {..in which case the long before the slice table has a -1 in the even word and the object size
001003           in the odd word, instead of a superlink}
```

```
001004
001005          objSize := pClasses^[index].objectSize;
001006          IF slices > 2 THEN
001007              BEGIN
001008              TPOctets(itsSTP)^[8]  := TPOctets(@objSize)^[0];
001009              TPOctets(itsSTP)^[12] := TPOctets(@objSize)^[1];
001010              itsSTP^[-1] := TPMethodArray(pSTables^[pClasses^[index].superIndex]);
001011              END
001012          ELSE
001013              BEGIN
001014              TPWords(itsSTP)^[-2] := -1;
001015              TPWords(itsSTP)^[-1] := objSize;
001016              END;
001017
001018          TPOctets(itsSTP)^[0] := TPOctets(@index)^[0];
001019          TPOctets(itsSTP)^[4] := TPOctets(@index)^[1];
001020          END;
001021
001022    {Report success to higher levels and let it copy the tables it may desire before we destroy them}
001023      CallFinishedProc(DefaultFinishedProc, 0);
001024      pClasses    := NIL;
001025      pSTables    := NIL;
001026      pAuthors    := NIL;
001027      pAliases    := NIL;
001028      pMethods    := NIL;
001029
001030    {Just to keep things clean and consistent}
001031      pHashName   := NIL;
001032      pHashUnit   := NIL;
001033
001034    {Disable UnitAuthor, ClassAuthor, ClassVersion, ClassSize, and FinishedProc}
001035      classesInitialized := TRUE;
001036
001037    {Exit from %_Pgm1, finally freeing its local storage below the TMethodArray}
001038      %_ExitPoppingTo(mAllocAddr);
001039  END;
001040
001041
001042  {Each unit ends with:
001043      .PROC unit#i
001044      JSR %_Unit      ; Defined below
001045      JSR unit#x      ; for every unit USEd by the unit within $CLASSES+ (in order USEd)...
001046      ...
001047      JSR unit#z
001048      JSR class-init#1  ; for every class implemented in unit#i...
001049      ...
001050      JSR class-init#k
001051      RTS             }
```

```
001052
001053  PROCEDURE %_Unit;
001054
001055      VAR unitPC:         LONGINT;
001056          hashUNIndex:    INTEGER;
001057
001058      FUNCTION CompareUnit(hashIndex: INTEGER): THashCompare;
001059          VAR pc:   LONGINT;
001060      BEGIN
001061          pc := pHashUnit^[hashIndex];
001062          IF pc = 0 THEN
001063              CompareUnit := cHole
001064          ELSE
001065          IF pc = unitPC THEN
001066              CompareUnit := cMatch
001067          ELSE
001068              CompareUnit := cMismatch;
001069      END;
001070
001071  BEGIN
001072      unitPC := CallPC;
001073      hashUNIndex := LookupInHashArray(maxUnits, unitPC, TRUE, CompareUnit);
001074      IF hashUNIndex > 0 THEN         {first time here -- let the initialization happen}
001075          pHashUnit^[hashUNIndex] := unitPC
001076      ELSE
001077          %_ExitCaller(0);            {exit from .PROC unit#i because we have already initialized this unit}
001078  END;
001079
001080
001081      {toInsert, return: -index if class already there or if table full, index (> 0) if a hole found}
001082      {not toInsert, return: index (> 0) if class found, -index if not there}
001083      {return 0 if table is full}
001084  FUNCTION LookupClassAlpha(keyA8: TA8; toInsert: BOOLEAN): INTEGER;
001085
001086      FUNCTION CompareName(hashIndex: INTEGER): THashCompare;
001087          VAR myIndex:    INTEGER;
001088      BEGIN
001089          myIndex := pHashName^[hashIndex];
001090          IF myIndex = 0 THEN
001091              CompareName := cHole
001092          ELSE
001093          IF pClasses^[myIndex].classAlpha = keyA8 THEN
001094              CompareName := cMatch
001095          ELSE
001096              CompareName := cMismatch;
001097      END;
001098
001099  BEGIN
```

```
001100      LookupClassAlpha := LookupInHashArray(maxClasses, ORD(keyA8[2])*ORD(keyA8[4])+ORD(keyA8[6]),
001101                                            toInsert, CompareName);
001102  END;
001103
001104
001105  FUNCTION FillArraysFrom(pc: LONGINT; itsLevelNumber: INTEGER; superSTP: TPSliceTable;
001106                          itsSTP: TPSliceTable; itsOddMethods: INTEGER): LONGINT;
001107          {** NO VAR PARAMETERS ALLOWED THAT ARE REFERENCED AFTER CALLING MAllocate **}
001108      VAR impLevelNumber:         INTEGER;
001109          impMethNumber:          INTEGER;
001110          targetLocation:         ProcPtr;
001111          fini:                   BOOLEAN;
001112          impMethodArrayPtr:      TPMethodArray;
001113          index:                  INTEGER;
001114          level:                  INTEGER;
001115          numAtThatLevel:         INTEGER;
001116          superMethodArrayPtr:    TPMethodArray;
001117          canInherit:             BOOLEAN;
001118          methodNumber:           INTEGER;
001119  BEGIN
001120      impLevelNumber := itsLevelNumber;
001121      impMethNumber := 0;
001122
001123      REPEAT
001124          targetLocation := %_NextMethod(pc, impLevelNumber, impMethNumber);
001125          fini := (targetLocation = p%_Class) OR (targetLocation = pJmp%_Class);
001126          IF NOT fini THEN
001127              BEGIN
001128              impMethodArrayPtr := itsSTP^[impLevelNumber];
001129              IF impMethodArrayPtr = NIL THEN
001130                  BEGIN
001131                  index := numClasses;
001132                  level := itsLevelNumber;    {always even}
001133                  {$IFC fTrcClascal}
001134                  WriteLn('pClasses = ', ORD(pClasses));
001135                  WriteLn('Index      Level',  '  impLevelNumber = ', impLevelNumber:3);
001136                  WriteLn(index:3, level:12);
001137                  {$ENDC}
001138                  WHILE level > impLevelNumber DO
001139                      BEGIN
001140                      index := pClasses^[index].superIndex;
001141                      level := level - 2;
001142                      {$IFC fTrcClascal}
001143                      WriteLn(index:3, level:12);
001144                      {$ENDC}
001145                      END;
001146
001147                  {$IFC fTrcClascal}
```

```
001148                  WriteLn('-- In FillArrays, making a new method table --');
001149                  WriteLn('pc = ', pc:12, '  itsLevelNumber = ', itsLevelNumber:3,
001150                          '  superSTP = ', ORD(superSTP):12, '  itsSTP = ', ORD(itsSTP):12);
001151                  WriteLn('  itsOddMethods = ', itsOddMethods:3,
001152                          '  impMethNumber = ', impMethNumber:3,
001153                          '  targetLocation = ', ORD(targetLocation):12);
001154                  WriteLn('  index = ', index:3,
001155                          '  level = ', level:3,
001156                          '  word[-2] = ', TPWords(pSTables^[index])^[-2]:7,
001157                          '  word[-1] = ', TPWords(pSTables^[index])^[-1]:7);
001158                  {$ENDC}
001159
001160                  numAtThatLevel := TPWords(pSTables^[index])^[ORD(ODD(impLevelNumber))-2];
001161
001162                  {$IFC fTrcClascal}
001163                  WriteLn('numAtThatLevel = ', numAtThatLevel:3);
001164                  {$ENDC}
001165
001166                  impMethodArrayPtr := TPMethodArray(MAllocate(numAtThatLevel, growMethods));
001167                  numMethods := numMethods + numAtThatLevel;
001168                  itsSTP^[impLevelNumber] := impMethodArrayPtr;
001169
001170                  IF superSTP = NIL THEN
001171                      superMethodArrayPtr := NIL
001172                  ELSE
001173                      superMethodArrayPtr := superSTP^[impLevelNumber]; {may be NIL}
001174
001175                  canInherit := (impLevelNumber < itsLevelNumber) AND (superMethodArrayPtr <> NIL);
001176
001177                  FOR methodNumber := 1 TO numAtThatLevel DO
001178                      IF canInherit THEN
001179                          impMethodArrayPtr^[methodNumber] := superMethodArrayPtr^[methodNumber]
001180                      ELSE
001181                          impMethodArrayPtr^[methodNumber] := ORD(@ _Abstract);
001182                  END;
001183              impMethodArrayPtr^[impMethNumber] := ORD(targetLocation);
001184              END;
001185      UNTIL fini;
001186
001187    {For any inherited slice that had no overrides, make it point at the same slice as the superclass}
001188      FOR level := 0 TO itsLevelNumber - 1 DO
001189          IF itsSTP^[level] = NIL THEN
001190              itsSTP^[level] := superSTP^[level]; {may be NIL, too}
001191
001192    {If the odd slice has only ABSTRACT methods, then use a global to tell %_Pgm2 what to do}
001193      IF itsSTP^[itsLevelNumber + 1] = NIL THEN
001194          IF itsOddMethods > biggestAbstractClass THEN
001195              biggestAbstractClass := itsOddMethods;
```

```
001196
001197      {$IFC fTrcClascal}
001198      DumpArrays;                {************************************************************************}
001199      {$ENDC}
001200
001201      FillArraysFrom := pc;
001202  END;
001203
001204
001205  {The class-init routine of TFoo = SUBCLASS OF TSuperclass starts with:
001206      JSR %_Class('TFOO    ', 'TSUPERCL', @sliceTable, sizeOfEvenSlice, sizeOfOddSlice, objSize); Defined below
001207      JSR method#1(sliceNumber*256 + methodNumber)    ; for every method in the IMPLEMENTATION
001208      ...                                             ; these calls are not executed: %_Class interprets them
001209      JSR method#r(sliceNumber*256 + methodNumber)    ; slice 0 is TObject, method 1 is first method
001210      JSR %_Class             ; just a terminator (The first call on %_Class interprets through here)}
001211
001212  PROCEDURE %_Class(itsClassName, itsSuperName: TS8; itsSTP: TPSliceTable;
001213                    itsEvenMethods, itsOddMethods, itsObjSize: INTEGER);
001214          {** NO VAR PARAMETERS ALLOWED THAT ARE REFERENCED AFTER CALLING RAllocate & FillArraysFrom **}
001215      VAR i:                  INTEGER;
001216          itsAlpha:           TA8;
001217          superAlpha:         TA8;
001218          nameHashIndex:      INTEGER;
001219          superClIndex:       INTEGER;
001220          superSTP:           TPSliceTable;
001221          itsLevelNumber:     INTEGER;
001222          pc:                 LONGINT;
001223          level:              INTEGER;
001224  BEGIN
001225    {First class of a unit?}
001226      IF CallCallPC <> currCallCallPC THEN
001227          BEGIN
001228          EndPreviousUnit;
001229          currCallCallPC := CallCallPC;
001230          END;
001231
001232    {Increment numClasses but first be sure there is room in the arrays... this could move ALL the arrays!}
001233      IF numClasses > (limClasses - 2) THEN
001234          BEGIN
001235          i {dummy}  := RAllocate(SIZEOF(TClassInfo), numClasses,
001236                                  growClasses, limClasses, maxClasses, 'Classes', ORD(pClasses));
001237          limClasses := RAllocate(SIZEOF(TPSliceTable), numClasses,
001238                                  growClasses, limClasses, maxClasses, 'Classes', ORD(pSTables));
001239          END;
001240      numClasses := numClasses + 1;
001241
001242    {Convert names from TS8 to TA8 type}
001243      FOR i := 1 TO 8 DO
```

```
001244          BEGIN
001245          itsAlpha[i] := itsClassName[i];
001246          superAlpha[i] := itsSuperName[i];
001247          END;
001248
001249      {Enter this class into the name hash table}
001250        nameHashIndex := LookupClassAlpha(itsAlpha, TRUE);   {Temporary variable needed because stack may quake}
001251        IF nameHashIndex > 0 THEN
001252            pHashName^[nameHashIndex] := numClasses
001253        ELSE
001254        {$IFC fDbgClascal}
001255        IF nameHashIndex < 0 THEN
001256            CLABreak('Class name appeared twice', numClasses)
001257        ELSE
001258            CLABreak('Class Name Hash table full', maxClasses);
001259        {$ELSEC}
001260            CLAFail(0);
001261        {$ENDC}
001262
001263      {Hash the name of the superclass}
001264        IF itsSuperName = 'NIL     ' THEN
001265            BEGIN           {This class has no superclass (e.g., TObject)}
001266            superClIndex := 0;
001267            itsLevelNumber := 0;
001268            superSTP := NIL;
001269            END
001270        ELSE
001271            BEGIN
001272            superClIndex := pHashName^[LookupClassAlpha(superAlpha, FALSE)];
001273            itsLevelNumber := NumSlices(superClIndex);
001274            superSTP := pSTables^[superClIndex];
001275            END;
001276
001277      {Fill this slice table with NILs for the moment}
001278        FOR level := 0 TO itsLevelNumber + 1 DO
001279            itsSTP^[level] := NIL;
001280
001281      {To be referenced from FillArraysFrom to calculate numAtThatLevel}
001282        TPWords(itsSTP)^[-2] := itsEvenMethods;
001283        TPWords(itsSTP)^[-1] := itsOddMethods;
001284
001285      {Initialize the fields of the class record}
001286        WITH pClasses^[numClasses] DO
001287            BEGIN
001288            classAlpha := itsAlpha;
001289            superIndex := superClIndex;
001290            objectSize := itsObjSize;       {may be changed by a call on  ClassSize from the class-init block}
001291            classAlias := 0;                {may be supplied by a call on ClassAuthor from the class-init block}
```

```
001292          companyAndAuthor := 0;          {may be supplied by a call on ClassAuthor or UnitAuthor}
001293          version := 1;                   {may be changed  by a call on ClassVersion from the class-init block}
001294          oldestReadableVersion := 1;     {may be changed  by a call on ClassVersion from the class-init block}
001295          END;
001296
001297    {Record the slice table pointer}
001298      pSTables^[numClasses] := itsSTP;
001299
001300    {Before running the user's class-init code, be sure there is space for him to add an Alias and two Authors}
001301      IF numAuthors > (limAuthors - 2) THEN
001302          limAuthors := RAllocate(SIZEOF(TA32), numAuthors,
001303                                  growAuthors, limAuthors, maxAuthors, 'Authors', ORD(pAuthors));
001304
001305      IF numAliases > (limAliases - 1) THEN
001306          limAliases := RAllocate(SIZEOF(TA8), numAliases,
001307                                  growAliases, limAliases, maxAliases, 'Aliases', ORD(pAliases));
001308
001309      {$IFC fTrcClascal}
001310      WriteLn(' End of %_Class!, stp = ', ORD(itsSTP):5);
001311      {$ENDC}
001312
001313    {[Interpret and] skip the MOVE/JSR pairs after the call of this procedure}
001314      SetCallPC(FillArraysFrom(CallPC, itsLevelNumber, superSTP, itsSTP, itsOddMethods));
001315  END;
001316
001317
001318  END.
001319

End of File -- Lines: 1319 Characters: 44959
```

```
================================================================================================
FILE: "LIBTK/UABC.TEXT"
================================================================================================

000001  {UNIT UABC}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004                      { *** METHODS NEED TO BE GROUPED INTO RIGHT CATEGORIES *** }
000005                      { *** ADD reserve IN ALMOST EVERY CLASS ***}
000006
000007  { UABC2.TEXT  TProcess-TDocDirectory-TDocManager-TClipboard-TCommand-TCutCopyCommand-TPasteCommand}
000008  { UABC3.TEXT  TImage-TView-TPaginatedView-TPageView-TPrintManager-THeading-TSelection}
000009  { UABC4.TEXT  TWindow-TDialogBox-TMenuBar-TFont}
000010  { UABC5.TEXT  TPanel-TBand-TPane-TMarginPad-TBodyPad-TScroller-TScrollBar}
000011
000012
000013  UNIT UABC;
000014
000015  {$SETC IsIntrinsic := TRUE }
000016
000017  {$IFC IsIntrinsic}
000018  INTRINSIC;
000019  {$ENDC}
000020
000021  INTERFACE
000022
000023
000024  USES
000025      {$U UnitStd     } UnitStd,      {Client should not USE UnitStd}
000026      {$U UnitHz      } UnitHz,       {Client should not USE UnitHz and MUST NOT USE Storage}
000027      {$U libtk/UObject     } UObject,      {Client must USE UObject}
000028      {$U -#BOOT-SysCall} SysCall,     {Client may USE SysCall}
000029  {$IFC LibraryVersion > 20}
000030      {$U LIBTK/Passwd} Passwd,
000031  {$ENDC}
000032  {$IFC LibraryVersion <= 20}
000033      {$U FontMgr     } FontMgr,      {Client should USE UFont instead of FontMgr before QuickDraw}
000034  {$ENDC}
000035      {$U QuickDraw   } QuickDraw,    {Client must USE QuickDraw (unless we provide a type-stub for it)}
000036  {$IFC LibraryVersion > 20}
000037      {$U FontMgr     } FontMgr,      {Client should USE UFont instead of FontMgr after QuickDraw}
000038  {$ENDC}
000039      {$U libtk/UDraw       } UDraw,        {Client must USE UDraw}
000040
000041                                      {Client need not USE anything below this line}
000042      {$U PMDecl      } PMDecl,
000043  {$IFC libraryVersion <= 20}       { P E P S I }
```

```
000044      {$U PrStd       } PrStd,
000045  {$ENDC}
000046      {$U WM.Events   } Events,
000047      {$U WM.Folders  } Folders,
000048      {$U WM.Menus    } Menus,
000049      {$U AlertMgr    } AlertMgr,
000050  {$IFC LibraryVersion <= 20}
000051      {$U PrProcs     } PrProcs,
000052  {$ENDC}
000053      {$U WMLstd      } WMLstd,
000054      {$U WMLCrs      } WMLCrs,
000055      {$U WMLSb       } WMLSb,
000056      {$U WMLGrow     } WMLGrow,
000057      {$U Scrap       } Scrap,
000058  {$IFC libraryVersion <= 20}
000059      {$U PrMgrUtil   } PrMgrUtil,
000060      {$U PrMgr       } PrMgr,
000061  {$ELSEC}                            { S P R I N G }
000062      {$U PrStdInfo}    PrStdInfo,
000063      {$U PrPublic}     PrPublic,
000064  {$ENDC}
000065      {$U FilerComm   } FilerComm;
000066
000067
000068  {$SETC fDbgABC     := fDbgOK}{FALSE}
000069  {$SETC fRngABC     := fDbgOK}{FALSE}
000070  {$SETC fSymABC     := fSymOK}{FALSE}
000071
000072  {$SETC fDebugMethods := fDbgABC} {if VAR also true, trace entries and/or exits}
000073
000074
000075  CONST
000076
000077      maxMenus        = 31;   {unfortunate, but menus must be in non-relocatable storage, & this is easiest}
000078      maxFonts        = 11;
000079      maxSegments     = 6;
000080      maxSegSize      = $20000; {128K}
000081      abortChunkSize  = 32768;  {32k}
000082
000083      iconNameSeparator = '<';{character separating parts of an icon name}
000084
000085      stdHHysteresis  = 9;    {amount the mouse must move from anchor before drag starts, unless}
000086      stdVHysteresis  = 6;    {           TSelection.GetHysteresis is overridden}
000087
000088      noCursor        = -2;   {                   used when you do not set the cursor}
000089      hiddenCursor    = -1;   {icrsHidden         Hides the cursor entirely}
000090      arrowCursor     =  1;   {icrsInactive       Standard arrow cursor}
000091      crossCursor     =  9;   {icrsLCCross        LisaCalc cross}
```

```
000092      textCursor      = 10;   {icrsXIBeam         Standard text I-Beam}
000093      checkCursor     = 12;   {icrsCheck          Checkmark}
000094      smCrossCursor   = 13;   {icrsGECross        LisaDraw cross (smaller than crossCursor)}
000095      fingerCursor    = 14;   {icrsLFinger        LisaDraw left-pointing finger}
000096
000097      firstUserCursor = 100;  { this is the smallest user-defined cursor }
000098
000099      nothingKind     = 0;
000100
000101      noCmdNumber     = 0;
000102
000103      docLdsn         = 3;        {ldsn for the first document data segment}
000104      docDsBytes      = 5120;     {default heap size for a document data segment}
000105      docExcess       = 2048;     {the virtual data segment may be this much larger than needed for the heap}
000106
000107      printLdsn       = 2;        {ldsn to hand to LisaPrint}
000108      ascArwDown      = $1F;
000109      ascArwLeft      = $1C;
000110      ascArwRight     = $1D;
000111      ascArwUp        = $1E;
000112      ascBackspace    = $08;
000113      ascClear        = $1B;
000114      ascEnter        = $03;
000115      ascReturn       = $0D;
000116      ascTab          = $09;
000117
000118    {alert phrase codes must be between 9 and 899}
000119
000120      phWordDelimiters=  9;
000121
000122      phTrouble       = 10;   {The tool is having trouble}
000123      phUnknown       = 11;   {Phrase(error) is undefined for this error}
000124      phNoText        = 21;
000125      phNoSel         = 22;
000126      phNoInsPt       = 23;
000127      phRevert        = 24;
000128      phRevBlank      = 25;
000129      phUnkCmd        = 26;
000130      phSelCant       = 27;
000131      phUnchanged     = 28;
000132      phSaving        = 29;
000133      phTerminated    = 30;
000134      phEditClip      = 31;
000135      phNoClip        = 32;
000136      phUnkClip       = 33;
000137      phDialogUp      = 34;
000138      phCantUndo      = 35;
000139      phNoCommand     = 36;
```

```
000140      phOlderVersion  = 37;
000141      phNewerVersion  = 38;
000142      phConverting    = 39;
000143      phAborting      = 40;
000144
000145      phPage          = 41; {+SW+}
000146      phTitle         = 42; {+SW+}
000147
000148      phCantSave      = 43;
000149      phCantRevert    = 44;
000150
000151      phCountry       = 45;
000152
000153
000154    {command, selection, and phrase indices used by Dialog Building Block}
000155      uCreateLayoutBox = 701;    {Command numbers}
000156      uMoveLayoutBoxes  = 702;
000157      uCmdLaunchHeading = 703;
000158      uCmdInstallMargins = 704;
000159
000160      layPickKind = 119;         {Selection kinds}
000161      layEditLegendKind = 133;
000162      frameKind = 161;
000163
000164      phTooManyChars = 101;      {Phrases}
000165      phOddEven = 102;
000166      phOddOnly = 103;
000167      phEvenOnly  = 104;
000168      phOddOrEven = 105;
000169      phMinPage = 106;
000170      phMaxPage = 107;
000171      phPageAlignment = 108;
000172      phAlignment = 109;
000173      phTopLeft = 110;
000174      phTopCenter = 111;
000175      phTopRight = 112;
000176      phBotLeft = 113;
000177      phBotCenter = 114;
000178      phBotRight = 115;
000179      phLaunchHeading = 116;
000180      phPageMargins = 117;
000181      phUnits = 118;
000182      phInches = 119;
000183      phCentimeters = 120;
000184      phLeft = 121;
000185      phLeftCluster = 122;
000186      phTop = 123;
000187      phTopCluster = 124;
```

```
000188      phRight = 125;
000189      phRightCluster = 126;
000190      phBottom = 127;
000191      phBotCluster = 128;
000192      phInstallMargins = 129;
000193      phInchTitle= 130;
000194      phCmTitle = 131;
000195      phNewHeading = 132; {+SW+}
000196
000197      phOK = 142;      {client should NOT lightly change the phrases for these, since they are used for}
000198      phCancel = 143;  {determining text and locations of OK/Cancel buttons for built-in ToolKit dialogs}
000199
000200      stdBoxWidth  = 17;  {dimensions for default checkboxes}
000201      stdBoxHeight = 11;
000202      stdBoxSpacing = 20;
000203
000204      stdCurvH  = 18;  {for Buttons}
000205      stdCurvV  = 14;
000206      stdBtnHeight = 22;
000207
000208      noIDNumber = -2;
000209      noId = '';
000210
000211      IDLength = 9; {the significant length of id strings}
000212
000213      stdTitleHeight = 10;     {for layout boxes}
000214      stdSlimTitleHeight = 6;
000215      stdLeftRightBorder = 3;
000216      stdBottomBorder = 2;
000217
000218    {errcodes of other libraries}
000219
000220      erAborted       = 4033; {user typed Apple-.; Desktop Manager}
000221      erDuplicateName =  890; {OS & Desktop Manager}
000222      erInvalidName   =  971; {OS & Desktop Manager}
000223      erNameNotFound  =  972; {OS & Desktop Manager}
000224
000225    {ToolKit errCodes must be between 4201 and 4499}
000226
000227      erPassword      = 4201;
000228      erVersion       = 4202;
000229      erBadData       = 4203;
000230      erCantRead      = 4304;
000231      erCantWrite     = 4305;
000232      erDirtyDoc      = 4306;
000233      erNoMoreDocs    = 4307;
000234      erNoMemory      = 4308;
000235      erNoDiskSpace   = 4309;
```

```
000236      erWrongPassword = 4310;
000237      erMaxToolKit    = 4499;
000238
000239    {command codes must be between 101 and 999}
000240
000241      uSetAllAside    = 101;
000242      uSetAside       = 102;
000243      uPutAway        = 103;
000244      uPrFmt          = 104;
000245      uPrintAsIs      = 111;
000246      uPrint          = 105;
000247      uPrMonitor      = 106;
000248      uSaveVersion    = 107;
000249      uRevertVersion  = 108;
000250      utSetAside      = 109;  {Set Aside ^Document^}
000251      uSetClipAside   = 110;
000252
000253      {Typing Buzzword}
000254      uTyping         = 150;
000255
000256      {The toolkit uses the following only as arguments to selection.CantDoCmd}
000257      uBackspace      = 151;
000258      uEnter          = 152;
000259      uForwardSpace   = 153;
000260      uReturn         = 154;
000261      uTab            = 155;
000262
000263      {The toolkit uses the following only as arguments to process.RememberCommand}
000264      uSomeCommand    = 156;
000265      uScrolling      = 157;
000266      uSplitting      = 158;
000267      uResizeWindow   = 159;
000268      uResizePanel    = 160;
000269      UMousePress     = 161;
000270      uThumbing       = 162;
000271      uMoveWindow     = 163;
000272      uKeyDown        = 164; {could be made the same as uTyping}
000273
000274      uCopy           = 201;
000275      uCut            = 202;
000276      uPaste          = 203;
000277      uSelAll         = 204;
000278      uUndoLast       = 205;
000279      utUndoLast      = 206;  {Undo ^Last Change^}
000280      utRedoLast      = 207;  {Redo ^Last Change^}
000281      uClear          = 208;
000282
000283  {$IFC LibraryVersion <= 20}
```

```
000284      uFnt0           = 300;
000285      uFnt1           = 301;
000286      uFnt2           = 302;
000287      uFnt3           = 303;
000288      uFnt4           = 304;
000289      uFnt5           = 305;
000290      uFnt6           = 306;
000291      uFnt7           = 307;
000292      uFnt8           = 308;
000293      uFnt9           = 309;
000294      uFnt10          = 310;
000295      uFnt11          = 311;
000296  {$ENDC}
000297      uModern         = 320 + famModern - famMin;     {should result in 320}
000298      uClassic        = 320 + famClassic - famMin;
000299
000300      u20Pitch        = 330 + size20Pitch - sizeMin;  {should result in 330}
000301      u15Pitch        = 330 + size15Pitch - sizeMin;
000302      u12Pitch        = 330 + size12Pitch - sizeMin;
000303      u10Pitch        = 330 + size10Pitch - sizeMin;
000304      u12Point        = 330 + size12Point - sizeMin;
000305      u14Point        = 330 + size14Point - sizeMin;
000306      u18Point        = 330 + size18Point - sizeMin;
000307      u24Point        = 330 + size24Point - sizeMin;
000308
000309      uPlain          = 351;
000310      uBold           = 352;
000311      uItalic         = 353;
000312      uUnderline      = 354;
000313      uShadow         = 355;
000314      uOutline        = 356;
000315      uSuperscript    = 357;
000316      uSubscript      = 358;
000317
000318      uPrvwMargins    = 401;
000319      uPrvwBreaks     = 402;
000320      uPrvwOff        = 403;
000321      uDesignPages    = 405;
000322
000323      uShowFullSize   = 406;
000324      uReduce70Pct    = 407;
000325      uReduceToFit    = 408;
000326
000327      uSetHorzBreak   = 411;
000328      uSetVertBreak   = 412;
000329      uClearBreaks    = 413;
000330
000331      uRiseVertically   = 421;
```

```
000332      uRiseHorizontally = 422;
000333
000334      uAddColumnStrip   = 431;
000335      uAddRowStrip      = 432;
000336
000337      uReportEvents   = 501;
000338
000339      uCountHeap      = 506;
000340
000341      uCheckIndices   = 509;
000342      uDumpGlobals    = 510;
000343      uDumpPrelude    = 511;
000344      uExperimenting  = 512;
000345      uReptGarbage    = 513;
000346      uFreeGarbage    = 514;
000347
000348      uMainScramble   = 515;
000349      uDocScramble    = 516;
000350
000351      uEditDialog     = 521;
000352      uStopEditDialog = 522;
000353
000354    { the standard WantMenu will return FALSE for any menus with menuID >= mBuzzword;
000355      buzzword menus should be assigned IDs >= 100;
000356      debug menus should be assigned IDs 90-99 }
000357
000358      {$IFC fDbgABC}
000359      mBuzzword       = 100;
000360      {$ELSEC}
000361      mBuzzword       =  90;
000362      {$ENDC}
000363
000364      mnuClipFilePrint = 1000;    {special menuID for Clipboard File/Print}
000365
000366      firstPrivateEvent = 100; {first event type that you can use in TProcess.SendEvent}
000367
000368      {$IFC NOT fDbgABC}
000369      fExperimenting  =   FALSE;  { not experimenting if debug code if off }
000370      {$ENDC}
000371
000372  TYPE
000373
000374      TPrinterMetrics = RECORD
000375          paperRect:     Rect;     {the physical rectangle}
000376          printRect:     Rect;     {the printable rectangle}
000377          res:           Point;    {resolution, spots/inch}
000378          reserve:       ARRAY[0..7] OF BYTE;
000379          END;
```

```
000380
000381      TPreviewMode = (mPrvwMargins, mPrvwBreaks, mPrvwOff);
000382
000383      TDiResponse = (diAccept, diDismissDialogBox, diGiveToMainWindow, diRefuse);
000384
000385      TEnumAbilities = (aBar, aScroll, aSplit);
000386      TAbilities = SET OF TEnumAbilities;                    {for TPanel.Divide/CREATE argument}
000387
000388      TUnitsFromEdge = (pixelsFromEdge, percentFromEdge);    {for TPanel.Divide argument}
000389
000390      TAlertArg = 1..5;
000391      TAlertCounter = 7..9;
000392
000393      TAlignment = (aLeft, aRight, aCenter, aJustify);
000394      TPageAlignment = (aTopLeft, aTopCenter, aTopRight, aBottomLeft, aBottomCenter, aBottomRight);
000395
000396      TClickState =  RECORD
000397                     where: Point;
000398                     when: LONGINT;
000399                     clickCount: INTEGER;
000400                     fShift, fOption, fApple: BOOLEAN;
000401                     END;
000402
000403      TCmdNumber = INTEGER;       {the unique identifier of a command in a menu (or elsewhere)}
000404
000405      TCmdPhase = (doPhase, undoPhase, redoPhase);{doPhase first time, then undoPhase & redoPhase alternately}
000406
000407      TCursorNumber = INTEGER;
000408
000409      TEnumIcons = (iSkewer, iScrollBack, iFlipBack, iGrayA, iThumb, iGrayB, iFlipFwd, iScrollFwd); {TIcon}
000410
000411      TMousePhase = (mPress, mMove, mRelease);
000412
000413      TRevelation = (revealNone, revealSome, revealAll);
000414
000415      TPrReserve = ARRAY [0..127] OF Byte;       {lengthened}
000416      TPrelude =
000417          RECORD
000418              password:       {2} INTEGER;
000419              version:        {2} INTEGER;        {*** Should also do ABC version protection***}
000420              country:        {2} INTEGER;
000421              language:       {2} INTEGER;
000422              preludeSize:    {2} INTEGER;        {SIZEOF(TPrelude), which precedes the heap}
000423              unused:         {6} ARRAY [0..5] OF Byte;
000424             {The above fields should occupy 16 bytes to meet the Lisa standard}
000425              printPref:    {128} TPrReserve;
000426              docSize:        {4} LONGINT;        {sum of the sizes of the consecutive data segments}
000427              numSegments:    {2} INTEGER;        {no. of segments; all but the last are maxSegSize bytes}
```

```
000428              docDirectory:   {4} TDocDirectory;  {whence one finds the class table and the window}
000429             {Other fields may be added later}
000430              END;
000431
000432      TPPrelude = ^TPrelude;
000433
000434      TSBoxID = LONGINT; {THSb alias}
000435
000436      TWindowID = LONGINT; {WindowPtr alias}
000437
000438      TWmgrCmd =
000439          RECORD
000440              cmdNumber:  INTEGER;                {the command number}
000441              menuIndex:  Byte;                   {the ordinal number of the menu in its menu bar (or file)}
000442              itemIndex:  Byte;                   {the ordinal number of the item in its menu}
000443          END;
000444
000445
000446      TProcess = SUBCLASS OF TObject {only one instance exists (process)}
000447
000448        {Variables}
000449
000450        {Creation/Destruction}
000451          FUNCTION  {TProcess.}CREATE(object: TObject; heap: THeap): TProcess;
000452
000453        {Debugging}
000454          {$IFC fDebugMethods}
000455          PROCEDURE {TProcess.}DontDebug;     {Turn off all debug flags when last document is closed}
000456          {$ENDC}
000457          {$IFC fDbgABC}
000458          PROCEDURE {TProcess.}DumpGlobals;   {Print most global variables on alternate screen}
000459          {$ENDC}
000460
000461
000462        {Cursor Tracking}
000463          PROCEDURE {TProcess.}ChangeCursor(cursorNumber: TCursorNumber);
000464                      { applications call ChangeCursor if they want to change the cursor shape }
000465          PROCEDURE {TProcess.}DoCursorChange(cursorNumber: TCursorNumber);
000466                      { applications implement DoCursorChange to test cursorNumber for one of their
000467                          cursor shapes; if found, it calls QuickDraw's SetCursor routine, otherwise
000468                          it calls the generic TProcess.DoCursorChange }
000469          PROCEDURE {TProcess.}TrackCursor;
000470
000471        {Error Reporting}
000472          PROCEDURE {TProcess.}ArgAlert(whichArg: TAlertArg; argText: S255); {whichArg = 1 to 5}
000473          FUNCTION  {TProcess.}Ask(phraseNumber: INTEGER): INTEGER;
000474          PROCEDURE {TProcess.}BeginWait(phraseNumber: INTEGER);
000475          FUNCTION  {TProcess.}Caution(phraseNumber: INTEGER): BOOLEAN;
```

```
000476          PROCEDURE {TProcess.}CountAlert(whichCtr: TAlertCounter; counter: INTEGER);
000477          PROCEDURE {TProcess.}DrawAlert(phraseNumber: INTEGER; marginLRect: LRect);
000478          PROCEDURE {TProcess.}EndWait;
000479          PROCEDURE {TProcess.}GetAlert(phraseNumber: INTEGER; VAR theText: S255);
000480          PROCEDURE {TProcess.}Note(phraseNumber: INTEGER);
000481          PROCEDURE {TProcess.}RememberCommand(cmdNumber: TCmdNumber);    { for ^C and ^K in alerts }
000482          FUNCTION  {TProcess.}Phrase(error: INTEGER): INTEGER;
000483          PROCEDURE {TProcess.}Stop(phraseNumber: INTEGER);
000484
000485        {Initiate/Terminate}
000486          PROCEDURE {TProcess.}Commence(phraseVersion: INTEGER); {process init after the process object exists}
000487          PROCEDURE {TProcess.}Complete(allIsWell: BOOLEAN);
000488
000489        {Abort Handling}
000490          FUNCTION  {TProcess.}AbortRequest: BOOLEAN;
000491          PROCEDURE {TProcess.}AbortXferSequential(whichWay: xReadWrite; pFirst: Ptr;
000492                                                   numBytes, chunkSize: LONGINT; fs: TFileScanner);
000493
000494        {Main Loop}
000495          PROCEDURE {TProcess.}ObeyEvents(FUNCTION StopCondition: BOOLEAN);
000496                      {This will return IF: (1) amDying is TRUE (application terminated)
000497                          or (2) StopCondition returns TRUE (StopCondition is checked
000498                          only when no events are available, before starting to idle.)}
000499
000500          PROCEDURE {TProcess.}ObeyFilerEvent;
000501          PROCEDURE {TProcess.}ObeyTheEvent;
000502          PROCEDURE {TProcess.}Run;
000503
000504        {Private Events (Inter-process communication)}
000505          PROCEDURE {TProcess.}HandlePrivateEvent(typeOfEvent: INTEGER; fromProcess: LONGINT;
000506                                                  when: LONGINT; otherData: LONGINT); DEFAULT;
000507          PROCEDURE {TProcess.}SendEvent(typeOfEvent: INTEGER; targetProcess: LONGINT; otherData: LONGINT);
000508
000509        {Memory Management}
000510          PROCEDURE {TProcess.}BindCurrentDocument;
000511
000512        {Open/Close Window/Document}
000513          FUNCTION  {TProcess.}NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
000514                                             : TDocManager; DEFAULT;
000515
000516        {External Document Support}
000517          PROCEDURE {TProcess.}CopyExternalDoc(VAR error: INTEGER;
000518                                 externalName, volumePrefix: TFilePath); DEFAULT;
000519              {This is called if the application puts icons into the clipboard and the user
000520                  then pastes them into a folder or disk.}
000521
000522          END;
000523
```

```
000524
000525      TDocDirectory = SUBCLASS OF TObject
000526
000527        {Variables}
000528          window:         TWindow;
000529          classWorld:     TClassWorld;
000530
000531        {Creation/Destruction}
000532          FUNCTION  {TDocDirectory.}CREATE(object: TObject; heap: THeap; itsWindow: TWindow;
000533                                           itsClassWorld: TClassWorld): TDocDirectory;
000534
000535        {Version Conversion}
000536          PROCEDURE {TDocDirectory.}Adopt;
000537
000538          END;
000539
000540
000541      TDocManager = SUBCLASS OF TObject
000542
000543        {Variables}
000544          files:
000545              RECORD
000546              volumePrefix:   TFilePath;              {Desktop Manager volume and prefix of OS files}
000547              volume:         TFilePath;              {Desktop Manager volume of OS files; -volname-}
000548  {$IFC LibraryVersion > 20}
000549              password:       TPassword;              {The password for this document}
000550  {$ENDC}
000551              saveExists:     BOOLEAN;                {whether Save file is known to exist and seem readable}
000552              shouldSuspend:  BOOLEAN;                {should we create suspend files?}
000553              shouldToolSave: BOOLEAN;                {should we create save files if opened as a tool?}
000554              END;
000555          dataSegment:
000556              RECORD
000557              refnum:         ARRAY [1..maxSegments] OF INTEGER;    {refnums of its data segments}
000558              preludePtr:     TPPrelude;              {a pointer to the prelude of the data segment}
000559              changes:        LONGINT;                {How many changes since the last checkpoint}
000560              END;
000561          docHeap:            THeap;                  {the heap starts after the prelude}
000562          window:             TWindow;                {the document's window (it is in the data segment)}
000563          pendingNote:        INTEGER;                {If <> 0, NOTE alert that was requested while inactive}
000564          openedAsTool:       BOOLEAN;
000565
000566        {Creation/Destruction}
000567          FUNCTION  {TDocManager.}CREATE(object: TObject; heap: THeap; itsPathPrefix: TFilePath): TDocManager;
000568
000569        {Debugging}
000570          {$IFC fDbgABC}
000571          PROCEDURE {TDocManager.}DumpPrelude;        {Print most of prelude on alternate screen}
```

```
000572          {$ENDC}
000573
000574        {Attributes}
000575          FUNCTION  {TDocManager.}WindowWithId(wmgrID: TWindowID): TWindow;
000576
000577        {Process Termination}
000578          PROCEDURE {TDocManager.}Complete(allIsWell: BOOLEAN);
000579
000580        {Open/Close Window}
000581          FUNCTION  {TDocManager.}NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; DEFAULT;
000582
000583        {Files}
000584          PROCEDURE {TDocManager.}Close(afterSuspend: BOOLEAN);
000585            { CloseFiles is for the application to override if it has any of its own files that must be
000586              closed }
000587          PROCEDURE {TDocManager.}CloseFiles;
000588          PROCEDURE {TDocManager.}Open(VAR error: INTEGER; wmgrID: TWindowID; VAR OpenedSuspended:Boolean);
000589          PROCEDURE {TDocManager.}OpenBlank(VAR error: INTEGER; wmgrID: TWindowID);
000590          PROCEDURE {TDocManager.}OpenSaved(VAR error: INTEGER; wmgrID: TWindowID);
000591          PROCEDURE {TDocManager.}OpenSuspended(VAR error: INTEGER; wmgrID: TWindowID);
000592          PROCEDURE {TDocManager.}RevertVersion(VAR error: INTEGER; wmgrID: TWindowID);
000593          PROCEDURE {TDocManager.}SaveVersion(VAR error: INTEGER; volumePrefix: TFilePath;
000594                                              andContinue: BOOLEAN);
000595          PROCEDURE {TDocManager.}Suspend(VAR error: INTEGER);
000596
000597        {Data Segment}
000598          PROCEDURE {TDocManager.}Assimilate(VAR error: INTEGER);
000599          PROCEDURE {TDocManager.}Bind; DEFAULT;
000600          PROCEDURE {TDocManager.}ConserveMemory(maxExcess: LONGINT; fGC: BOOLEAN);
000601                                       {if fGC is TRUE also do a garbage collect -- on debugging versions,
000602                                           we just report garbage, on non-debugging versions we free it
000603                                           also.}
000604          PROCEDURE {TDocManager.}Deactivate;
000605          FUNCTION  {TDocManager.}DfltHeapSize: LONGINT;
000606          PROCEDURE {TDocManager.}ExpandMemory(bytesNeeded: LONGINT);
000607          PROCEDURE {TDocManager.}KillSegments(first, last: INTEGER);
000608          PROCEDURE {TDocManager.}MakeSegments(VAR error: INTEGER; oldSegments: INTEGER; newDocSize: LONGINT);
000609          PROCEDURE {TDocManager.}ResumeAfterOpen(VAR error: INTEGER; wmgrID: TWindowID);
000610          PROCEDURE {TDocManager.}SetSegSize(VAR error: INTEGER; minSize, maxExcess: LONGINT);
000611          PROCEDURE {TDocManager.}Unbind; DEFAULT;
000612          END;
000613
000614      TClipboard = SUBCLASS OF TDocManager
000615
000616        {Variables}
000617          hasView:            BOOLEAN;        {FALSE if no tool-kit-specific representation available}
000618          hasPicture:         BOOLEAN;        {FALSE if no universal picture available}
000619          hasUniversalText:   BOOLEAN;        {FALSE if no universal text available}
```

```
000620          hasIcon:            BOOLEAN;        {TRUE if there is an icon reference available}
000621        {****NOTE: The only way into or out of Universal Text is via the Universal Text Building Block****}
000622          cuttingTool:        LONGINT;        {The tool number of the tool that loaded the Clipboard, or 0}
000623          cuttingProcessID:   LONGINT;        {The OS process ID of the tool that loaded the Clipboard, or 0}
000624          clipCopy:           TFileScanner;   {IF <> NIL a scanner on the file containing a copy of the
000625                                                  clipboard before conversion.}
000626
000627         {Creation/Destruction}
000628          FUNCTION  {TClipboard.}CREATE(object: TObject; heap: THeap): TClipboard;
000629
000630         {Editing}
000631          PROCEDURE {TClipboard.}AboutToCut;      {whether or not data will actually be put in the data seg}
000632          PROCEDURE {TClipboard.}BeginCut;
000633          PROCEDURE {TClipboard.}EndCut;
000634
000635         {Undo}
000636          PROCEDURE {TClipboard.}CommitCut;
000637          FUNCTION  {TClipboard.}UndoCut: BOOLEAN; {return TRUE if succeeds}
000638
000639         {Identification}
000640          PROCEDURE {TClipboard.}Inspect;
000641          PROCEDURE {TClipboard.}Publicize;
000642
000643         {Data Segment}
000644          {PROCEDURE  TClipboard. Bind;}
000645          {PROCEDURE  TClipboard. Unbind;}
000646
000647          END;
000648
000649
000650      TCommand = SUBCLASS OF TObject
000651
000652         {Variables}
000653          cmdNumber:      TCmdNumber;     {the command number of the menu item that describes the command;
000654                                          usually the same one the user chose, but not necessarily}
000655          image:          TImage;         {If non-NIL, affects filtering by image.EachVirtualPart}
000656          undoable:       BOOLEAN;        {TRUE iff this command is undoable}
000657          doing:          BOOLEAN;        {TRUE if Performing or just did doPhase or redoPhase}
000658          revelation:     TRevelation;    {revealNone/Some/All of selection before performing command}
000659          unHiliteBefore: ARRAY [TCmdPhase] OF BOOLEAN; {TRUE -> Toolkit unhilites all selections before
000660                                                          perform}
000661          hiliteAfter:    ARRAY [TCmdPhase] OF BOOLEAN; {TRUE -> Toolkit hilites all selections after perform}
000662
000663         {Creation/Destruction}
000664          FUNCTION  {TCommand.}CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
000665                                  itsImage: TImage; isUndoable: BOOLEAN; itsRevelation: TRevelation): TCommand;
000666
000667         {Filtering}
```

```
000668          PROCEDURE {TCommand.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
000669          PROCEDURE {TCommand.}FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
000670
000671        {Command Execution}
000672          PROCEDURE {TCommand.}Commit; DEFAULT;                          {commit a command}
000673          PROCEDURE {TCommand.}Perform(cmdPhase: TCmdPhase); DEFAULT;  {do, undo, or redo a command}
000674
000675          END;
000676
000677
000678      TCutCopyCommand = SUBCLASS OF TCommand
000679
000680        {Variables}
000681          isCut: BOOLEAN;        {TRUE iff this was a cut; FALSE iff a copy}
000682
000683        {Creation/Destruction}
000684          FUNCTION  {TCutCopyCommand.}CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
000685                                        itsImage: TImage; isCutCmd: BOOLEAN): TCutCopyCommand;
000686
000687        {Command Execution}
000688         {PROCEDURE  TCutCopyCommand. Commit;}
000689          PROCEDURE {TCutCopyCommand.}DoCutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN;
000690                                                cmdPhase: TCmdPhase); DEFAULT;
000691                          {the clipboard is already set up; you only have to load data into it in doPhase}
000692         {PROCEDURE  TCutCopyCommand. Perform(cmdPhase: TCmdPhase);}
000693
000694          END;
000695
000696
000697      TPasteCommand = SUBCLASS OF TCommand
000698
000699        {Creation/Destruction}
000700          FUNCTION  {TPasteCommand.}CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
000701                                        itsImage: TImage): TPasteCommand;
000702
000703        {Command Execution}
000704          PROCEDURE {TPasteCommand.}DoPaste(clipSelection: TSelection; pic: PicHandle;
000705                                         cmdPhase: TCmdPhase); DEFAULT;
000706                          {the clipboard is already set up, except in undoPhase sel & pic are NIL}
000707         {PROCEDURE  TPasteCommand. Perform(cmdPhase: TCmdPhase);}
000708
000709          END;
000710
000711
000712      TImage = SUBCLASS OF TObject
000713
000714        {Variables}
000715          extentLRect:        LRect;          {the bounding box for updates; also for default hit-testing}
```

```
000716          view:               TView;
000717          allowMouseOutside:  BOOLEAN;        {If TRUE, TImage.MouseTrack will NOT force the mouse point
000718                                                  to lie within the extentLRect; TImage.CREATE sets this FALSE}
000719
000720      {methods}
000721          FUNCTION  {TImage.}CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsView: TView): TImage;
000722
000723          FUNCTION  {TImage.}CursorAt(mouseLPt: lPoint): TCursorNumber; DEFAULT;
000724          PROCEDURE {TImage.}Draw; DEFAULT;
000725          PROCEDURE {TImage.}EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); DEFAULT;
000726          PROCEDURE {TImage.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); DEFAULT;
000727          PROCEDURE {TImage.}FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
000728          PROCEDURE {TImage.}HaveView(view: TView); DEFAULT;
000729          FUNCTION  {TImage.}Hit(mouseLPt: lPoint): BOOLEAN; DEFAULT;
000730          PROCEDURE {TImage.}Invalidate;  {does NOT do it on all pads}
000731          FUNCTION  {TImage.}LaunchLayoutBox(view: TView): TImage; DEFAULT;
000732          PROCEDURE {TImage.}OffSetBy(deltaLPt: LPoint); DEFAULT;
000733          PROCEDURE {TImage.}OffSetTo(newTopLeft: LPoint);
000734          PROCEDURE {TImage.}MouseMove(mouseLPt: lPoint); DEFAULT;
000735          PROCEDURE {TImage.}MousePress(mouseLPt: lPoint); DEFAULT;
000736          PROCEDURE {TImage.}MouseRelease; DEFAULT;
000737          PROCEDURE {TImage.}MouseTrack(mPhase: TMousePhase; mouseLPt: LPoint); DEFAULT;
000738          PROCEDURE {TImage.}ReactToPrinterChange; DEFAULT;
000739          PROCEDURE {TImage.}RecalcExtent; DEFAULT;
000740          PROCEDURE {TImage.}Resize(newExtent: LRect); DEFAULT;
000741          FUNCTION  {TImage.}SeesSameAs(image: TImage): BOOLEAN; DEFAULT; {$}
000742
000743          END;
000744
000745
000746      TView = SUBCLASS OF TImage
000747
000748        {Variables}
000749          panel:              TPanel;                 {The panel in which it is viewed}
000750          clickLPt:           LPoint;                 {The last place the user clicked the mouse button}
000751          printManager:       TPrintManager;          {NIL if view not printable}
000752          res:                Point;                  {resolution, spots/inch}
000753
000754          screenPad:          TPad;       {like noPad, but scales from view coords to screen coords if view
000755                                           resolution and screen resolution differ
000756                                           *** CAUTION -- Only for mapping coordinates-- DO NOT try to
000757                                           Focus this pad or do Invals, etx ***}
000758
000759          fitPagesPerfectly:  BOOLEAN;  {whether view size should fluctuate automatically so that one always
000760                                           ends up with an even number of pages}
000761
000762          isPrintable:        BOOLEAN;                {Whether this view can be printed}
000763          isMainView:         BOOLEAN;     {FALSE if an auxiliary view, such as page view or paginated view}
```

```
000764          stdScroll:          LPoint;
000765          scrollPastEnd:      Point;                {Amount we should scroll past the end of the view}
000766
000767
000768        {Creation/Destruction}
000769          FUNCTION  {TView.}CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsExtent: LRect;
000770              itsPrintManager: TPrintManager; itsDfltMargins: LRect; itsFitPagesPerfectly:BOOLEAN;
000771              itsRes: Point; isMainView: BOOLEAN): TView;
000772           {PROCEDURE  TView. Free;}
000773
000774        {Attributes}
000775          PROCEDURE {TView.}BeInPanel(panel: TPanel);
000776          PROCEDURE {TView.}GetStdScroll(VAR deltaLStd: LPoint);
000777          FUNCTION  {TView.}MaxPageToPrint: LONGINT;
000778
000779        {Pagination}
000780          PROCEDURE {TView.}AddStripOfPages(vhs: VHSelect); DEFAULT;
000781          FUNCTION  {TView.}ForceBreakAt(vhs: VHSelect; precedingLocation: LONGINT;
000782              proposedLocation: LONGINT): LONGINT;
000783          PROCEDURE {TView.}RedoBreaks; DEFAULT;
000784          PROCEDURE {TView.}RemapManualBreaks(
000785               FUNCTION NewBreakLocation(vhs: VHSelect; oldBreak: LONGINT): LONGINT);
000786
000787        {Cross-Panel Drag}
000788          FUNCTION  {TView.}DoReceive(selection: TSelection; lPtInView: LPoint): BOOLEAN;
000789
000790        {Direct Display Permission -- per panel}
000791          FUNCTION  {TView.}OKToDrawIn(lRectInView: LRect): BOOLEAN;  {Default is FALSE; app can override}
000792
000793        {Cursor tracking - per pane}
000794          {FUNCTION  TView. CursorAt(mouseLPt: LPoint): TCursorNumber;}
000795
000796        {Resizing}
000797           {PROCEDURE  TView. Resize(newExtent: LRect);}
000798          PROCEDURE {TView.}SetMinViewSize(VAR minLRect: LRect);
000799
000800        {Clipboard Setup}
000801          PROCEDURE {TView.}CreateUniversalText;
000802
000803        {Variables embedded in text}
000804          PROCEDURE {TView.}SetFunctionValue(keyword: S255; VAR itsValue: S255);
000805
000806        {Selecting}
000807          FUNCTION  {TView.}NoSelection: TSelection;
000808          END;
000809
000810
000811      TPaginatedView = SUBCLASS OF TView
```

```
000812
000813        {Variables}
000814          unpaginatedView:    TView;             {the unpaginated view from whence this derives}
000815
000816          pageSize:   ARRAY[VHSelect] OF LONGINT;  {width/height of a page's representation on the screen,
000817                                                    in the same metrics as the regular view -- could still
000818                                                    differ from actual screen space a/c screen horiz/vertical
000819                                                    resolution}
000820
000821          workingInMargins:   BOOLEAN;
000822
000823        {Creation/Destruction}
000824          FUNCTION  {TPaginatedView.}CREATE(object: TObject; heap: THeap;
000825                                        itsUnpaginatedView: TView): TPaginatedView;
000826
000827         {PROCEDURE  TPaginatedView. AddStripOfPages(vhs: VHSelect);}
000828          PROCEDURE {TPaginatedView.}AdornPageOnScreen;
000829         {FUNCTION   TPaginatedView. CursorAt(mouseLPt: LPoint): TCursorNumber;}
000830          PROCEDURE {TPaginatedView.}DepagifyLPoint(pagLPt: LPoint; VAR unPagLPt: LPoint);
000831          PROCEDURE {TPaginatedView.}DoOnPages(focusOnInterior: BOOLEAN;  PROCEDURE DoOnAPage);
000832         {PROCEDURE  TPaginatedView. Draw;}
000833         {PROCEDURE  TPaginatedView. MouseTrack(mPhase: TPhase; mouseLPt: LPoint);}
000834          PROCEDURE {TPaginatedView.}PagifyLPoint(unPagLPt: LPoint; VAR pagLPt: LPoint);
000835         {PROCEDURE  TPaginatedView. ReactToPrinterChange;}
000836         {PROCEDURE  TPaginatedView. RedoBreaks;}
000837
000838        END;
000839
000840
000841      TPageView = SUBCLASS OF TView
000842
000843          FUNCTION  {TPageView.}CREATE(object: TObject; heap: THeap;
000844                                    itsPrintManager: TPrintManager): TPageView;
000845         {PROCEDURE  TPageView. Draw;}
000846          END;
000847
000848
000849      THeading = SUBCLASS OF TImage      {a header/footer image}
000850
000851          printManager:           TPrintManager;
000852          pageAlignment:          TPageAlignment;
000853          offsetFromAlignment:    LPoint;
000854
000855          oddOnly:        BOOLEAN;   {to restrict printing only to odd-numbered pages}
000856          evenOnly:       BOOLEAN;   { ditto even }
000857          minPage:        LONGINT;   {minimum page number to want this heading}
000858          maxPage:        LONGINT;   {maximum page number to want it}
000859
```

```
000860        {Creation/Destruction}
000861          FUNCTION  {THeading.}CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
000862                                    itsExtentLRect: LRect; itsPageAlignment: TPageAlignment;
000863                                    itsOffsetFromAlignment: LPoint): THeading;
000864
000865        {Attributes}
000866          PROCEDURE {THeading.}ChangePageAlignment(newPageAlignment: TPageAlignment);
000867
000868        {Selective Use}
000869          FUNCTION  {THeading.}ShouldDraw(pageNumber: LONGINT): BOOLEAN;
000870          FUNCTION  {THeading.}ShouldFrame: BOOLEAN; DEFAULT;
000871
000872        {Display}
000873          PROCEDURE {THeading.}AdjustForPage(pageNumber: LONGINT; editing: BOOLEAN); DEFAULT;
000874          PROCEDURE {THeading.}LocateOnPage(editing: BOOLEAN);
000875         {PROCEDURE  THeading. Draw;}
000876          END;
000877
000878
000879      TPrintManager = SUBCLASS OF TObject
000880          view:               TView;
000881          pageView:           TView;
000882
000883          breaks:             ARRAY[VHSelect] OF TArray; {of LONGINT}
000884                                  {pagebreak representation: absolute value gives location; negative
000885                                  signifies manual break; nonnegative signifies automatic pagebreak}
000886
000887          pageMargins:        LRect;     {in view resolution; top and left are > 0 , bot & right < 0}
000888
000889          headings:           TList; {OF THeading}
000890
000891          canEditPages:       BOOLEAN;
000892          layoutDialogBox:    TDialogBox;
000893
000894          frameBody:          BOOLEAN;
000895          paperLRect:         LRect;
000896          printableLRect:     LRect;
000897
000898          contentLRect:       LRect;  {the inner rectangle into which chunks of view are stuffed}
000899
000900          printerMetrics:     TPrinterMetrics;    {physical properties of the printer}
000901
000902          pageRiseDirection:  VHSelect;
000903               {if 'h', it means that page numbers rise from left to right fastest;
000904                if 'v', it means that page numbers rise from top to bottom fastest;
000905                default value is 'h'}
000906
000907          FUNCTION  {TPrintManager.}CREATE(object: TObject; heap: THeap): TPrintManager;
```

```
000908          PROCEDURE {TPrintManager.}Init(itsMainView: TView; itsDfltMargins: LRect);
000909         {PROCEDURE  TPrintmanager. Free;}
000910
000911          PROCEDURE {TPrintManager.}AddStripOfPages(vhs: VHSelect);
000912          PROCEDURE {TPrintManager.}ChangeMargins(margins: LRect);
000913          PROCEDURE {TPrintManager.}ClearPageBreaks(automatic: BOOLEAN);
000914          PROCEDURE {TPrintManager.}DrawBreaks(manualOnly:  BOOLEAN);
000915          PROCEDURE {TPrintManager.}DrawOneBreak(pageBreak: LONGINT; vhs: vhSelect);
000916          PROCEDURE {TPrintManager.}DrawPage;
000917          PROCEDURE {TPrintManager.}EnterPageEditing;
000918          PROCEDURE {TPrintManager.}GetPageLimits(pageNumber: LONGINT; VAR viewLRect: LRect);
000919          FUNCTION  {TPrintManager.}NewPaginatedView(object: TObject): TPaginatedView;
000920          FUNCTION  {TPrintManager.}NewPageView(object: TObject): TView;
000921          FUNCTION  {TPrintManager.}PageWith(VAR lPtInView: LPoint; VAR strip: Point): LONGINT;
000922          PROCEDURE {TPrintManager.}Print(printPref: TPrReserve);
000923          PROCEDURE {TPrintManager.}ReactToPrinterChange;
000924          PROCEDURE {TPrintManager.}RedoBreaks;
000925          PROCEDURE {TPrintManager.}SetBreak(vhs: VHSelect; where: LONGINT; isAutomatic: BOOLEAN);
000926          PROCEDURE {TPrintManager.}SetDfltHeadings; DEFAULT;
000927          PROCEDURE {TPrintManager.}SkipPage(pageNumber: LONGINT);
000928
000929      END; {TPrintManager definition}
000930
000931
000932      TSelection = SUBCLASS OF TObject
000933
000934        {Variables}
000935          window:         TWindow;            {the window in which it was made}
000936          panel:          TPanel;             {the panel in which it was made}
000937          view:           TView;              {the view or subview of panel in which it was made}
000938          kind:           INTEGER;            {0 means no selection, rest of codes are defined by view}
000939          anchorLPt:      LPoint;             {the place the mouse went down (view-relative)}
000940          currLPt:        LPoint;             {the place the mouse was last tracked}
000941          boundLRect:     LRect;              {bounding box of the selection} {+++LSR+++}
000942          coSelection:    TSelection;         {if non-NIL, a selection to forward unimplemented methods to}
000943          canCrossPanels: BOOLEAN;            {:=TRUE in MousePress/FALSE in MouseRelease for cross-panel drag}
000944
000945        {Creation/Destruction}
000946          FUNCTION  {TSelection.}CREATE(object: TObject; heap: THeap; itsView: TView; itsKind: INTEGER;
000947                                        itsAnchorLPt: LPoint): TSelection;
000948         {FUNCTION   TSelection. Clone(heap: THeap): TObject;}    {clones coSelection}
000949          FUNCTION  {TSelection.}FreedAndReplacedBy(selection: TSelection): TSelection;
000950
000951        {Attributes}
000952          PROCEDURE {TSelection.}GetHysteresis(VAR hysterPt: Point); DEFAULT; {rtns a delta from orig panel pt}
000953          PROCEDURE {TSelection.}HaveView(view: TView);
000954
000955        {Files}
```

```
000956          PROCEDURE {TSelection.}MarkChanged; DEFAULT;                        {Increment change counters}
000957
000958        {Command Dispatch}
000959          FUNCTION  {TSelection.}CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; DEFAULT;
000960          PROCEDURE {TSelection.}CantDoCmd(cmdNumber: TCmdNumber); DEFAULT;
000961          PROCEDURE {TSelection.}CantDoIt; DEFAULT;
000962          PROCEDURE {TSelection.}DoKey(ascii: CHAR; keycap: Byte; shiftKey, appleKey, optionKey: BOOLEAN);
000963          FUNCTION  {TSelection.}NewCommand(cmdNumber: TCmdNumber): TCommand; DEFAULT;
000964          PROCEDURE {TSelection.}PerformCommand(command: TCommand; cmdPhase: TCmdPhase); DEFAULT;
000965
000966        {Idle}
000967          PROCEDURE {TSelection.}IdleBegin(centiSeconds: LONGINT); DEFAULT;
000968          PROCEDURE {TSelection.}IdleContinue(centiSeconds: LONGINT); DEFAULT;
000969          PROCEDURE {TSelection.}IdleEnd(centiSeconds: LONGINT); DEFAULT;
000970
000971        {Editing -- to be overridden by applications}
000972          PROCEDURE {TSelection.}KeyBack(fWord: BOOLEAN); DEFAULT;
000973          PROCEDURE {TSelection.}KeyChar(ch: CHAR); DEFAULT;
000974          PROCEDURE {TSelection.}KeyClear; DEFAULT;
000975          PROCEDURE {TSelection.}KeyEnter(dh, dv: INTEGER); DEFAULT;
000976          PROCEDURE {TSelection.}KeyForward(fWord: BOOLEAN); DEFAULT;
000977          PROCEDURE {TSelection.}KeyPause; DEFAULT; {Pause in typing}
000978          PROCEDURE {TSelection.}KeyReturn; DEFAULT;
000979          PROCEDURE {TSelection.}KeyTab(fBackward: BOOLEAN); DEFAULT;
000980          PROCEDURE {TSelection.}SelectParagraphs;
000981
000982        {Drawing -- per pane}
000983          PROCEDURE {TSelection.}Highlight(highTransit: THighTransit); DEFAULT;
000984
000985        {Selecting}
000986          PROCEDURE {TSelection.}DeSelect; DEFAULT;
000987          PROCEDURE {TSelection.}DrawGhost; DEFAULT;
000988          PROCEDURE {TSelection.}MousePress(mouseLPt: LPoint); DEFAULT;
000989          PROCEDURE {TSelection.}MouseMove(mouseLPt: LPoint); DEFAULT;
000990          PROCEDURE {TSelection.}MouseRelease; DEFAULT;
000991          PROCEDURE {TSelection.}MoveBackToAnchor; DEFAULT;   {called when cross-panel drag has been refused}
000992
000993        {Undo Maintenance}
000994          PROCEDURE {TSelection.}Restore; DEFAULT;
000995          PROCEDURE {TSelection.}Save; DEFAULT;
000996
000997        {Scroll into view}
000998          PROCEDURE {TSelection.}Reveal(asMuchAsPossible: BOOLEAN); DEFAULT;
000999
001000          END;
001001
001002
001003      TWindow = SUBCLASS OF TArea
```

```
001004
001005        {Variables}
001006          panels:         TList {OF TPanel};          {The panels in the window (at least one)}
001007          panelTree:      TArea;                      {no panels: NIL, one panel: that; else a TBranchArea}
001008          dialogBox:      TDialogBox;                 {NIL if SELF IS a dialog box window}
001009          selectPanel:    TPanel;                     {The panel with the active selection}
001010          undoSelPanel:   TPanel;                     {The selectPanel during the last command}
001011          clickPanel:     TPanel;                     {The panel in which the user last clicked in a pane}
001012          undoClickPanel: TPanel;                     {The clickPanel during the last command}
001013          selectWindow:   TWindow;                    {The window with the active selection -- either
001014                                                         SELF or its Dialogbox }
001015          undoSelWindow:  TWindow;                    {the selectWindow during the last command}
001016          wmgrID:         TWindowID;                  {ORD(Pointer to the Window Manager's GrafPort)}
001017          isResizable:    BOOLEAN;                    {Is there a Resize Box}
001018          believeWmgr:    BOOLEAN;                    {TRUE iff the Toolkit should believe the window
001019                                                          manager's idea of the size of the window;
001020                                                          this will be FALSE (for example) if we create
001021                                                          the window object before the window is put on
001022                                                          the screen.}
001023          maxInnerSize:   Point;                      {The window size the user explicitly set with grow
001024                                                          icon}
001025          changes:        LONGINT;                    {How many changes since the last save}
001026          lastCmd:        TCommand;                   {last undoable command object}
001027          printerMetrics: TPrinterMetrics;            {Properties of the printer currently formatted for}
001028          pgSzOK:         BOOLEAN;                    {Whether to allow user-defined page-sizes in Fmt For
001029                                                          Printer dialog}
001030          pgRgOK:         BOOLEAN;                    {Whether page-range dialog should be enabled in PRINT...
001031                                                           dialog -- normally TRUE}
001032          panelToPrint:   TPanel;                     {NB: IF >1 printable panel in window, choice should be
001033                                                           made by providing separate menu items}
001034          objectToFree:   TObject;                    {used to stash a reference to an object which should be
001035                                                          freed at end of event loop}
001036        {Creation/Destruction}
001037          FUNCTION  {TWindow.}CREATE(object: TObject; heap: THeap; itsWmgrID: TWindowID; itsResizability
001038                                  : BOOLEAN): TWindow;
001039         {PROCEDURE  TWindow. Free;}
001040
001041        {$IFC fDbgABC}
001042        {Debugging}
001043          PROCEDURE {TWindow.}ToggleFlag(VAR flag: BOOLEAN); DEFAULT; {Toggle a debug flag in a menu}
001044         {$ENDC}
001045
001046        {Attributes}
001047         {PROCEDURE  TWindow. GetMinExtent(VAR minExtent: Point; windowIsResizingIt: BOOLEAN);}
001048          PROCEDURE {TWindow.}GetTitle(VAR title: S255); {Get the window title}
001049          FUNCTION  {TWindow.}IsActive: BOOLEAN;
001050          FUNCTION  {TWindow.}IsVisible: BOOLEAN;
001051          PROCEDURE {TWindow.}SetWmgrId(itsWmgrId: TWindowID); {Also sets port fields of panes}
```

```
001052
001053        {Buttoning}
001054          PROCEDURE {TWindow.}DownEventAt(mousePt: Point); DEFAULT;
001055         {FUNCTION   TWindow. DownAt(mousePt: Point): BOOLEAN;}
001056
001057        {Dialog Box affairs}
001058          PROCEDURE {TWindow.}PutUpDialogBox(dialogBox: TDialogBox); DEFAULT;
001059          PROCEDURE {TWindow.}TakeDownDialogBox; DEFAULT;
001060
001061        {Display}
001062         {PROCEDURE  TWindow. Focus;}
001063         {PROCEDURE  TWindow. Frame;}
001064          PROCEDURE {TWindow.}Highlight(highTransit: THighTransit); DEFAULT;
001065         {PROCEDURE  TWindow. Refresh(rActions: TActions; highTransit: THighTransit);}
001066          PROCEDURE {TWindow.}Update(doHilite: BOOLEAN); DEFAULT;
001067
001068        {Resizing}
001069          PROCEDURE {TWindow.}DownInSizeBox(mousePt: Point); DEFAULT;
001070          PROCEDURE {TWindow.}Resize(moving: BOOLEAN); DEFAULT; {Reset size from portRect size (w. adjustments)}
001071          PROCEDURE {TWindow.}ResizeTo(newSize: Point); DEFAULT; {callable from application}
001072
001073        {Command Dispatch and Menus}
001074          FUNCTION  {TWindow.}CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; DEFAULT;
001075          FUNCTION  {TWindow.}CanDoStdCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; DEFAULT;
001076          PROCEDURE {TWindow.}CommitLast; DEFAULT;
001077          PROCEDURE {TWindow.}DoCommand(cmdNumber: TCmdNumber); DEFAULT;
001078          PROCEDURE {TWindow.}LoadMenuBar; DEFAULT;
001079          PROCEDURE {TWindow.}MenuEventAt(mousePt: Point); DEFAULT;
001080          FUNCTION  {TWindow.}NewCommand(cmdNumber: TCmdNumber): TCommand; DEFAULT;
001081          FUNCTION  {TWindow.}NewStdCommand(cmdNumber: TCmdNumber): TCommand;
001082          PROCEDURE {TWindow.}PerformCommand(newCommand: TCommand);
001083          PROCEDURE {TWindow.}PerformLast(cmdPhase: TCmdPhase);
001084          PROCEDURE {TWindow.}SaveCommand(command: TCommand); {NOTE: do not use the arg after calling this;
001085                                                               use window.lastCmd instead}
001086          PROCEDURE {TWindow.}SetupMenus;
001087          PROCEDURE {TWindow.}UndoLast;
001088          FUNCTION  {TWindow.}WantMenu(menuID: INTEGER; inClipboard: BOOLEAN): BOOLEAN;
001089
001090        {Miscellaneous}
001091          PROCEDURE {TWindow.}AbortEvent; {only QuickPort should override this}
001092
001093        {Selection Maintenance during commands}
001094          PROCEDURE {TWindow.}RestoreSelection;
001095          PROCEDURE {TWindow.}RevealSelection(asMuchAsPossible, doHilite: BOOLEAN);
001096          PROCEDURE {TWindow.}SaveSelection;
001097
001098        {Desktop}
001099              {The following 2 methods assume that we are focused on the window before they are called}
```

```
001100          PROCEDURE {TWindow.}Activate;
001101          PROCEDURE {TWindow.}Deactivate;
001102          PROCEDURE {TWindow.}BlankStationery; DEFAULT;
001103          PROCEDURE {TWindow.}StashPicture(highTransit: THighTransit);
001104
001105  {$IFC LibraryVersion > 20}
001106        {Desktop Manager Communication}
001107          PROCEDURE {TWindow.}NameToPrefix(VAR error, offset: INTEGER; VAR name, prefix: TFilePath);
001108          PROCEDURE {TWindow.}PrefixToName(VAR error, offset: INTEGER; VAR prefix, name: TFilePath);
001109
001110                (*Convert between OS prefix (ie., '-volname-{DxxxTyyy}' and an icon pathname (ie.,
001111                     '<diskname<foldername1<foldername2<...<iconname').  If an error is returned,
001112                     offset will point just beyond the part of the name that caused the error, e.g.
001113                     if '<office<forms<expenses' returns erDuplicateName and the offset is 14
001114                     (pointing to the third '<') then there is more than one 'forms' folder on the
001115                     office disk.  Error constants are defined above.
001116
001117                   NOTE: these methods will likely take a while to execute, since the Desktop Manager
001118                   must be swapped in to process the request.  Therefore, you should try to minimize the
001119                   number of times these are called.*)
001120  {$ENDC}
001121
001122        {Foci of Attention}
001123          FUNCTION  {TWindow.}CursorFeedback: TCursorNumber;
001124          PROCEDURE {TWindow.}PickStdCursor;
001125
001126        {Printing}
001127          PROCEDURE {TWindow.}AcceptNewPrintingInfo(document: TDocManager; prReserve: TPrReserve);
001128          PROCEDURE {TWindow.}ChkPrMismatch;
001129          PROCEDURE {TWindow.}GetPrinterMetrics;
001130          PROCEDURE {TWindow.}Print(panel: TPanel; nixPgRange: BOOLEAN; nixWholeDialog: BOOLEAN);
001131
001132        {Filtering}
001133          PROCEDURE {TWindow.}EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); {For app to implement}
001134          PROCEDURE {TWindow.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
001135          PROCEDURE {TWindow.}FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
001136          PROCEDURE {TWindow.}FilterDispatch(actualObj: TObject; image: TImage;
001137                                             PROCEDURE DoToObject(filteredObj: TObject));
001138
001139        {Idle}
001140          PROCEDURE {TWindow.}IdleBegin(centiSeconds: LONGINT);
001141          PROCEDURE {TWindow.}IdleContinue(centiSeconds: LONGINT);
001142          PROCEDURE {TWindow.}IdleEnd(centiSeconds: LONGINT);
001143
001144          END;
001145
001146
001147      TDialogBox = SUBCLASS OF TWindow
```

```
001148
001149        {Variables}
001150          keyResponse:                TDiResponse;
001151          menuResponse:               TDiResponse;
001152          downInMainWindowResponse:   TDiResponse;
001153
001154          freeOnDismissal:            BOOLEAN;
001155
001156        {Creation/Destruction}
001157          FUNCTION  {TDialogBox.}CREATE(object: TObject; heap: THeap; itsResizability: BOOLEAN;
001158                                        itsHeight: INTEGER;  itsKeyResponse, itsMenuResponse,
001159                                        itsDownInMainWindowResponse: TDiResponse): TDialogBox;
001160
001161        {Attributes}
001162         {PROCEDURE  TDialogBox. GetMinExtent(VAR minExtent: Point; windowIsResizingIt: BOOLEAN);}
001163
001164        {Display}
001165          PROCEDURE {TDialogBox.}Appear;
001166          PROCEDURE {TDialogBox.}BeDismissed; DEFAULT;
001167          PROCEDURE {TDialogBox.}Disappear; DEFAULT;
001168
001169          END;
001170
001171
001172      TBand = SUBCLASS OF TArea
001173
001174        {Variables}
001175          window:         TWindow;
001176          panes:          TList {OF TPane};
001177          panel:          TPanel;
001178          scroller:       TScroller;                  {the scroll box}
001179          scrollDir:      VHSelect;                   {v if a row of panes with a vertical bar,
001180                                                       h if a column of panes with a horizontal bar}
001181
001182        {Creation/Destruction}
001183          FUNCTION  {TBand.}CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsInnerRect: Rect;
001184                                   itsScroller: TScroller; itsDir: VHSelect): TBand;
001185         {PROCEDURE  TBand. Free;}
001186
001187        {Attributes}
001188          FUNCTION  {TBand.}ViewLCd: LONGINT;
001189
001190        {Scrolling}
001191          PROCEDURE {TBand.}OffsetPanes(deltaLPt: LPoint);
001192          PROCEDURE {TBand.}ScrollBy(deltaLCd: LONGINT);
001193                          {A TBand can only scroll in one direction; this also moves the thumb}
001194          PROCEDURE {TBand.}ScrollStep(icon: TEnumIcons; deltaLStd: LONGINT);
001195          PROCEDURE {TBand.}ScrollTo(viewLCd: LONGINT);
```

```
001196          FUNCTION  {TBand.}ThumbPos: INTEGER;
001197          PROCEDURE {TBand.}ThumbTo(newThumbPos: INTEGER);
001198
001199
001200        {Resizing}
001201         {PROCEDURE  TBand. ResizeOutside(newOuterRect: Rect);}
001202          PROCEDURE {TBand.}ResizePanes(newViewLCd: LONGINT);
001203
001204          END;
001205
001206
001207      TSideBand = SUBCLASS OF TBand
001208        {Fields}
001209          topOrLeft:  BOOLEAN;
001210          {NOTE: SELF.scroller is NIL}
001211
001212          FUNCTION  {TSideBand.}CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsInnerRect: Rect;
001213                              itsDir: VHSelect; itsTopOrLeft: BOOLEAN;
001214                              itsViewLCd: LONGINT): TSideBand;
001215
001216        {Attributes}
001217          FUNCTION  {TSideBand.}CoBand: TBand;
001218              {returns the band adjacent to SELF}
001219          END;
001220
001221
001222      TPanel = SUBCLASS OF TArea
001223
001224        {Variables}                                    {panes are listed row-wise in the panes list}
001225          window:              TWindow;
001226          panes:               TList {OF TPane};
001227          currentView:         TView;                          {The view seen through SELF: normal or paginated}
001228          view:                TView;                          {The unpaginated view seen through SELF }
001229          paginatedView:       TPaginatedView;                 {NIL if not previewing margins}
001230          selection:           TSelection;                     {the current selection}
001231          undoSelection:       TSelection;                     {the selection to be restored for an undo/redo}
001232          bands:               ARRAY[VHSelect] OF TList;       {redundant... bands[v].at(1) = top row of panes}
001233          scrollBars:          ARRAY[VHSelect] OF TScrollBar;  {scrollBars[(v,h)]--the (vert,horiz) scroll bars}
001234          abilities:           ARRAY[VHSelect] OF TAbilities;  {[aBar, aScroll, aSplit, aResize]}
001235          minInnerDiagonal:    Point;
001236          resizeBranch:        TBranchArea;                    {the branch that my botRight resizes, or NIL}
001237          zoomed:              BOOLEAN;
001238          zoomFactor:          TScaler;
001239          previewMode:         TPreviewMode;
001240          lastClick:           RECORD                          {describes the pane the user last clicked}
001241                               CASE gotPane: BOOLEAN OF
001242                                   TRUE: (clickPane: TPane);   {the last pane the user clicked on}
001243                                   FALSE: (clickPt: Point);    {the innerRect.topLeft of lastClick.pane,
```

```
001244                                                          in the case where the lastClick.pane was
001245                                                          deleted}
001246                              END;
001247          contentRect:        Rect;                           {part of the innerRect not containing side bands}
001248          tlSideBandSize:     Point;  {size of topLeft side bands}
001249          brSideBandSize:     Point;
001250              {NOTE: The sideband sizes refer to the size of the innerRect of the side band;
001251                     therefore a size of -1 means there is no side band on that side}
001252          deletedSplits:      TArray; {If NIL, don't remember splits that go away because the panel
001253                                       shrinks.  Otherwise, this should be a TArray with recordBytes
001254                                       2.  This is initialized to NIL in TPanel.CREATE; clients can
001255                                       allocate an array and change the field if they desire.}
001256
001257        {Creation/Destruction}
001258          FUNCTION  {TPanel.}CREATE(object: TObject; heap: THeap; itsWindow: TWindow;
001259                                    minHeight, minWidth: INTEGER; itsVAbilities, itsHAbilities: TAbilities): TPanel;
001260         {PROCEDURE  TPanel. Free;}
001261          PROCEDURE {TPanel.}HaveView(view: TView);
001262          FUNCTION  {TPanel.}NewView(object: TObject; itsExtent: LRect; itsPrintManager: TPrintManager;
001263                                     itsDfltMargins: LRect; itsFitPerfectlyOnPages: BOOLEAN): TView;
001264          FUNCTION  {TPanel.}NewStatusView(object: TObject; itsExtent: LRect): TView;
001265
001266        {Attributes}
001267          PROCEDURE {TPanel.}ComputeContentRect;
001268          PROCEDURE {TPanel.}DecideAboutBars(newOuterRect: Rect); {Decide if to have scroll bars & resize icon}
001269         {PROCEDURE  TPanel. GetMinExtent(VAR minExtent: Point; windowIsResizingIt: BOOLEAN);}
001270         {PROCEDURE  TPanel. GetBorder(VAR border: Rect);}
001271          FUNCTION  {TPanel.}FindBranchThatIsResized: TBranchArea;
001272          FUNCTION  {TPanel.}PaneShowing(anLRect: LRect): TPane;  {Returns first pane showing an part
001273                                                                  of anLRect, else NIL}
001274          PROCEDURE {TPanel.}SetInnerRect(newInnerRect: Rect); OVERRIDE;
001275          PROCEDURE {TPanel.}SetOuterRect(newOuterRect: Rect); OVERRIDE;
001276
001277        {Paneling the window}
001278          FUNCTION  {TPanel.}Divide(vhs: VHSelect;
001279                                    fromEdgeOfPanel: INTEGER; units: TUnitsFromEdge;
001280                                    whoCanResizeIt: TResizability;
001281                                    minSize: INTEGER; itsVAbilities, itsHAbilities: TAbilities): TPanel;
001282          PROCEDURE {TPanel.}Insert(panel: TPanel; vhs: VHSelect;
001283                                    fromEdgeOfPanel: INTEGER; units: TUnitsFromEdge;
001284                                    whoCanResizeIt: TResizability);          {Resizes both to share my space}
001285          PROCEDURE {TPanel.}Remove;                  {Does not Free SELF; Expands sibling to fill my space}
001286          PROCEDURE {TPanel.}Replace(panel: TPanel);  {Does not Free SELF; Resizes panel to fit my old space}
001287
001288        {Buttoning}
001289         {FUNCTION   TPanel. DownAt(mousePt: Point): BOOLEAN;}
001290          PROCEDURE {TPanel.}DownInSizeBox(mousePt: Point);
001291          PROCEDURE {TPanel.}HitScroller(vhs: VHSelect; mousePt: Point; scroller: TScroller; icon: TEnumIcons);
```

```
001292
001293        {Selecting}
001294          PROCEDURE {TPanel.}BeginSelection;
001295          PROCEDURE {TPanel.}BeSelectPanel(inSelectWindow: BOOLEAN);
001296    (*    FUNCTION  {TPanel.}NoSelection: TSelection; *)
001297
001298        {Cursor tracking}
001299          FUNCTION  {TPanel.}CursorAt(mousePt: Point): TCursorNumber;
001300
001301        {Display}
001302         {PROCEDURE  TPanel. Frame;}
001303          PROCEDURE {TPanel.}Highlight(selection: TSelection; highTransit: THighTransit);
001304                                { this highlights the selection on all pads }
001305          PROCEDURE {TPanel.}Invalidate;
001306                                { this invalidates the whole panel }
001307          PROCEDURE {TPanel.}InvalLRect(lRectInView: LRect);
001308                                { this invalidates the given LRect on all pads }
001309          FUNCTION  {TPanel.}OKToDrawIn(lRectInView: LRect): BOOLEAN;
001310                                { If this returns FALSE, commands must InvalLRect or XOR, not Draw or Erase }
001311          PROCEDURE {TPanel.}OnAllPadsDo(PROCEDURE DoOnThePad);
001312         {PROCEDURE  TPanel. Refresh(rActions: TActions; highTransit: THighTransit);}
001313          PROCEDURE {TPanel.}Rescroll;
001314          PROCEDURE {TPanel.}SetZoomFactor(zoomNumerator, zoomDenominator: Point);
001315
001316        {Page-Previewing}
001317          PROCEDURE {TPanel.}Preview(newMode: TPreviewMode);
001318
001319        {Printing}
001320          PROCEDURE {TPanel.}PrintView(printPref: TPrReserve);
001321
001322        {Scrolling}
001323          PROCEDURE {TPanel.}AutoScroll(mousePt: Point);
001324          PROCEDURE {TPanel.}DoScrolling(inArea: TArea; itsPane: TPane;
001325                                  hOk, vOk: BOOLEAN; VAR deltaLPt: LPoint);
001326                      {inArea must be a TBand or a TPane; if a TPane then inArea=itsPane;
001327                       if a TBand then itsPane is any one of the band's panes}
001328          FUNCTION  {TPanel.}PaneToScroll(VAR anLRect: LRect; hMinToSee, vMinToSee: INTEGER): TPane;
001329                      {Returns the pane to scroll for showing the minimum desired part ofLRect;
001330                          if that part is already showing, it returns NIL;
001331                      NOTE: anLRect is NOT changed}
001332          PROCEDURE {TPanel.}RevealLRect(VAR anLRect: LRect; hMinToSee, vMinToSee: INTEGER);
001333                      {Show at least the desired part of the LRect in the pane returned by PaneToShow;
001334                      NOTE: anLRect is NOT changed}
001335
001336        {Splitting}
001337          PROCEDURE {TPanel.}CleanUpPanes(deleteList: TList);
001338          PROCEDURE {TPanel.}MakeBand(vhs: VHSelect; scroller, prevScroller: TScroller);
001339          PROCEDURE {TPanel.}MoveSplitBefore(scroller: TScroller; newSkwrCd: INTEGER);
```

```
001340          FUNCTION  {TPanel.}NewBand(heap: THeap; myInnerRect: Rect;
001341                                    scroller: TScroller; vhs: VHSelect): TBand;
001342          FUNCTION  {TPanel.}NewPane(heap: THeap; innerRect: Rect; viewedLRect: LRect): TPane;
001343          PROCEDURE {TPanel.}RemakePanes;
001344          PROCEDURE {TPanel.}RememberSplit(vhs: VHSelect; atCd: INTEGER);
001345          PROCEDURE {TPanel.}RepaneOrthogonalBands(vhs: VHSelect);
001346          PROCEDURE {TPanel.}RestoreSplits;
001347
001348      {Side Bands}
001349          PROCEDURE {TPanel.}ShowSideBand(vhs: VHSelect; topOrLeft: BOOLEAN; size: INTEGER; viewLCd: LONGINT);
001350          PROCEDURE {TPanel.}SideBandRect(vhs: VHSelect; topOrLeft: BOOLEAN; VAR bandRect: Rect);
001351                  {returns the innerRect of the side band, given SELF.contentRect}
001352
001353        {Resizing}
001354          PROCEDURE {TPanel.}ResizeBand(vhs: VHSelect; band: TBand; newViewLCd: LONGINT;
001355                                        fInvalidate: BOOLEAN);
001356         {PROCEDURE  TPanel. ResizeInside(newInnerRect: Rect);}
001357         {PROCEDURE  TPanel. ResizeOutside(newOuterRect: Rect);}
001358
001359          END;
001360
001361
001362      TPane = SUBCLASS OF TPad
001363
001364        {Variables}
001365          currentView:    TView;                      {The view that is currently}
001366          panel:          TPanel;                     {The containing panel}
001367
001368        {Creation/Destruction}
001369          FUNCTION  {TPane.}CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsInnerRect: Rect;
001370                                   itsViewedLRect: LRect): TPane;
001371          PROCEDURE {TPane.}HaveView(view: TView);
001372
001373        {Attributes}
001374          PROCEDURE {TPane.}GetScrollLimits(VAR viewedLRect, scrollableLRect: LRect);
001375         {PROCEDURE  TPane. SetZoomFactor(zoomNumerator, zoomDenominator: Point);}
001376
001377        {Selecting}
001378          PROCEDURE {TPane.}MouseTrack(mPhase: TMousePhase; mousePt: Point);
001379                              {assumes mousePt is in the pane's innerRect}
001380
001381        {Cursor tracking}
001382          FUNCTION  {TPane.}CursorAt(mousePt: Point): TCursorNumber;
001383
001384        {Display}
001385         {PROCEDURE  TPane. Refresh(rActions: TActions; highTransit: THighTransit);}
001386
001387        {Resizing}
```

```
001388          PROCEDURE {TPane.}Resize(newInnerRect: Rect; vhs: VHSelect);
001389
001390        {Scrolling}
001391          PROCEDURE {TPane.}ScrollBy(VAR deltaLPt: LPoint);
001392                          {NOTE: deltaLPt is NOT changed; also moves the thumb(s)}
001393          PROCEDURE {TPane.}ScrollToReveal(VAR anLRect: LRect; hMinToSee, vMinToSee: INTEGER);
001394                          {NOTE: anLRect is NOT changed}
001395          END;
001396
001397
001398      TMarginPad = SUBCLASS OF TPad
001399
001400        {Variables}
001401          view:               TView;    {The view seen on the BODY of this page}
001402          pageNumber:         LONGINT;
001403          bodyPad:            TBodyPad;
001404
001405        {Creation/Destruction}
001406          FUNCTION {TMarginPad.}CREATE(object: TObject; heap: THeap): TMarginPad;
001407
001408          PROCEDURE {TMarginPad.}Rework(itsView: TView; itsOrigin: Point; itsRes: Point;
001409                        itsPageNumber: LONGINT; itsZoomFactor: TScaler; itsPort: GrafPtr);
001410          PROCEDURE {TMarginPad.}SetForPage(itsPageNumber: LONGINT; itsOrigin: Point);
001411
001412        {Display}
001413         {PROCEDURE  TMarginPad. Focus;}
001414
001415        {Process termination and Debugging Assistance}
001416         {PROCEDURE TMarginPad. Crash;}
001417         {FUNCTION  TMarginPad. BindHeap(activeVsClip, doBind: BOOLEAN): THeap;}
001418
001419          END;
001420
001421
001422      TBodyPad = SUBCLASS OF TPad
001423
001424        {Variables}
001425          marginPad:   TMarginPad; {the page shell whose body I am}
001426          nonNullBody:  Rect;  {the portion of the pad in the range of the mapped view;
001427            BodyPad.innerRect = nonNullBody unless manual pagebreak or end-of-view forces
001428            a shortage of view to map into entire inner rect} {someday make this comment comprehensible}
001429
001430        {Creation/Destruction}
001431          FUNCTION  {TBodyPad.}CREATE(object: TObject; heap: THeap; itsMarginPad: TMarginPad): TBodyPad;
001432          PROCEDURE {TBodyPad.}Recompute;
001433          PROCEDURE {TBodyPad.}SetForPage(itsPageNumber: LONGINT);
001434
001435        {Display}
```

```
001436          {PROCEDURE  TBodyPad. Focus;}
001437
001438        END;
001439
001440
001441      TScroller = SUBCLASS OF TObject
001442
001443        {Variables}
001444          scrollBar:      TScrollBar;                    {the scroll bar of which it is part}
001445          band:           TBand;                         {the object that can respond to scroll events}
001446          sBoxID:         TSBoxID;                       {the scroll-bar-library representation}
001447
001448        {Creation/Destruction}
001449          FUNCTION  {TScroller.}CREATE(object: TObject; heap: THeap; itsScrollBar: TScrollBar; itsId: TSBoxID)
001450                                    : TScroller;
001451          {PROCEDURE  TScroller. Free;}
001452
001453        {Attributes}
001454          PROCEDURE {TScroller.}GetSize(VAR boxRect: Rect);
001455          FUNCTION  {TScroller.}ScrollDir: VHSelect;
001456          PROCEDURE {TScroller.}SetSize(ownerRect: Rect);
001457          FUNCTION  {TScroller.}ThumbRange: INTEGER;
001458
001459        {Buttoning}
001460          PROCEDURE {TScroller.}TrackSkewer(mousePt: Point; VAR newSkwrCd: INTEGER;
001461                                            VAR scroller, prevScroller: TScroller);
001462          PROCEDURE {TScroller.}TrackThumb(mousePt: Point; VAR oldThumbPos, newThumbPos: INTEGER);
001463
001464        {Display}
001465          PROCEDURE {TScroller.}FillIcon(icon: TEnumIcons; fBlack: BOOLEAN);
001466          PROCEDURE {TScroller.}MoveThumb(newThumbPos: INTEGER);
001467
001468        {Splitting}
001469          PROCEDURE {TScroller.}ResplitAt(newSkwrCd: INTEGER; prevScroller: TScroller);
001470          PROCEDURE {TScroller.}SplitAt(newSkwrCd: INTEGER; VAR nextScroller: TScroller);
001471
001472          END;
001473
001474
001475      TScrollBar = SUBCLASS OF TObject
001476
001477        {Variables}
001478          firstBox:   TScroller;                         {the rest are found via the SB Library}
001479          isVisible:  BOOLEAN;                           {TRUE iff this scroll bar should be drawn}
001480
001481        {Creation/Destruction}
001482          FUNCTION  {TScrollBar.}CREATE(object: TObject; heap: THeap; vhs: VHSelect; outerRect: Rect;
001483                                        itsVisibility: BOOLEAN): TScrollBar;
```

```
001484          PROCEDURE {TScrollBar.}ChangeVisibility(needsBothBars: BOOLEAN;
001485                                                  bandOuterRect: Rect; itsAbilities: TAbilities);
001486
001487        {Buttoning}
001488          FUNCTION  {TScrollBar.}DownAt(mousePt: Point; VAR scroller: TScroller; VAR icon: TEnumIcons): BOOLEAN;
001489
001490        {Display}
001491          PROCEDURE {TScrollBar.}Draw;
001492          PROCEDURE {TScrollBar.}Erase;
001493
001494          END;
001495
001496
001497      TMenuBar = SUBCLASS OF TObject {only one instance exists (menuBar)}
001498
001499        {Variables}
001500          isLoaded:       ARRAY [1..maxMenus] OF BOOLEAN;     {TRUE iff the i'th menu has been inserted}
001501          mapping:        TArray {OF TWmgrCmd};               {maps command number to menu & item indices}
001502          numMenus:       INTEGER;                            {how many menus}
001503          numCommands:    INTEGER;                            {how many commands in all menus together}
001504
001505        {Creation/Destruction}
001506          FUNCTION  {TMenuBar.}CREATE(object: TObject; heap: THeap; itsScanner: TFileScanner): TMenuBar;
001507
001508        {Attributes}
001509          PROCEDURE {TMenuBar.}Check(cmdNumber: TCmdNumber; checked: BOOLEAN);
001510          PROCEDURE {TMenuBar.}Enable(cmdNumber: TCmdNumber; canBeChosen: BOOLEAN);
001511          PROCEDURE {TMenuBar.}BuildCmdName(destCmd, templateCmd: TCmdNumber; param: TPString);
001512                      {if param is NIL, use the default}
001513          FUNCTION  {TMenuBar.}GetCmdName(cmdNumber: TCmdNumber; pName: TPString): BOOLEAN;
001514                      {returns TRUE iff cmdNumber is found (pName will be empty);
001515                          pName can be NIL, which will save the overhead of returning the
001516                          menu item, for case where you just want to see if it exists}
001517          PROCEDURE {TMenuBar.}PutCmdName(cmdNumber: TCmdNumber; pName: TPString);
001518
001519        {Buttoning}
001520          FUNCTION  {TMenuBar.}CmdKey(ch: CHAR): TCmdNumber;
001521          FUNCTION  {TMenuBar.}DownAt(mousePt: Point): TCmdNumber;
001522
001523        {Display}
001524          PROCEDURE {TMenuBar.}Draw;
001525          PROCEDURE {TMenuBar.}EndCmd;
001526          PROCEDURE {TMenuBar.}HighlightMenu(withCmd: TCmdNumber);
001527                  {call this when the user presses the CLEAR key for example, to highlight
001528                      the appropriate menu title; you should then call window.DoCommand with
001529                      an apropriate command number.}
001530
001531        {Loading}
```

```
001532          PROCEDURE {TMenuBar.}Delete(menuID: INTEGER);
001533          PROCEDURE {TMenuBar.}Insert(menuID, beforeId: INTEGER);
001534          PROCEDURE {TMenuBar.}Unload;
001535
001536        {For Future Use}
001537          FUNCTION  {TMenuBar.}MenuWithID(menuID: INTEGER): Ptr;
001538          END;
001539
001540
001541  {$IFC LibraryVersion <= 20 AND FALSE} {do it this way in case we need it back for Pepsi version}
001542      TFont = SUBCLASS OF TObject
001543
001544        {Variables}
001545          family:         INTEGER;                        {Font Manager TFam}
001546
001547        {Creation/Destruction}
001548          FUNCTION  {TFont.}CREATE(object: TObject; heap: THeap; itsFamily: INTEGER): TFont;
001549
001550          END;
001551  {$ENDC}
001552
001553      { GLOBAL VARIABLES -- EFFECTIVELY, FIELDS OF CLASS TProcess }
001554
001555  VAR
001556
001557      activeWindowID:     TWindowID;                {The wmgrID field of the active document, or 0}
001558      allowAbort:         BOOLEAN;                  {Iff TRUE, allow aborts}
001559      autoBreakPen:       PenState;                 {pen to use to draw automatic page breaks}
001560      blinkOffCentiSecs:  LONGINT;                  {Centiseconds to hide the insertion point}
001561      blinkOnCentiSecs:   LONGINT;                  {Centiseconds to display the insertion point}
001562      boundClipboard:     TClipboard;               {The clipboard whose data segment is bound, or NIL}
001563      boundDocument:      TDocManager;              {The document whose data segment is bound, or NIL}
001564      cancelString:       STRING[20];               {The word "Cancel" for use in buttons}
001565      clickState:         TClickState;              {Shifts and repeats of the last mouse click}
001566      clipboard:          TClipboard;               {The Clipboard document manager}
001567      clipPrintPref:      TPrReserve;               {the print-preference for the clipboard}
001568      closedBySuspend:    BOOLEAN;                  {Iff TRUE, closedDocument was just suspended}
001569      closedDocument:     TDocManager;              {If not NIL, this document was just put away}
001570      cornerNumberStyle:  TTypeStyle;               {TypeStyle used for page-numbers in page-preview}
001571      countryCode:        INTEGER;                  {The country code as read from phrase file}
001572      currentDocument:    TDocManager;              {The active document OR if running in background, the
001573                                                        document to use; otherwise NIL}
001574      currentWindow:      TWindow;                  {currentDocument.window, OR NIL}
001575      cursorShape:        TCursorNumber;            {The cursor shape as recorded by TProcess.ChangeCursor}
001576      deferUpdate:        BOOLEAN;                  {set TRUE by app to defer updating while typing}
001577      dfltNewHeading:     STRING[20]; {+SW+}        {Default value for newly-created headings}
001578      docList:            TList {OF TDocManager};   {The documents that are open}
001579      eventTime:          LONGINT;                  {The time of the most recent WM event}
```

```
001580      eventType:          INTEGER;                    {The type number of the most recent WM event}
001581  {$IFC fDbgABC}
001582      fExperimenting:     BOOLEAN;                    {IF TRUE, enable zoom experimentation etc.}
001583      fCountHeap:         BOOLEAN;                    {Iff TRUE and IFC fCheckHeap, count objects once per cmd}
001584  {$ENDC}
001585  {$IFC LibraryVersion <= 20 AND FALSE} {do it this way in case we need it back for the Pepsi version}
001586      fonts:              ARRAY [0..maxFonts] OF TFont;
001587  {$ENDC}
001588      genClipPic:         BOOLEAN;                    {Iff TRUE, we are generating the Clipboard picture}
001589      highLevel:          ARRAY [BOOLEAN] OF THighTransit;    {TRUE=>hOffToOn, FALSE=>hOffToDim}
001590      highToggle:         ARRAY [BOOLEAN] OF THighTransit;    {TRUE=>hOffToOn, FALSE=>hOnToOff}
001591      idleTime:           LONGINT;                    {The time we finished processing the last user input}
001592      inBackground:       BOOLEAN;                    {Iff TRUE, currently running in background}
001593      limboPen:           PenState;                   {pen to use to fill limbo area in paginated view}
001594      manualBreakPen:     PenState;                   {pen to use to draw manual page breaks}
001595      marginPattern:      LPattern;                   {pattern to use to fill margins in paginated view}
001596      menuBar:            TMenuBar;                   {The menus of the application and the Clipboard}
001597      myProcessID:        LONGINT;                    {The OS ID of this process}
001598      myTool:             LONGINT;                    {The tool number of this tool}
001599      normalPen:          PenState;                   {pen state resulting from PenNormal}
001600      okString:           STRING[20];                 {The word "OK" for use in buttons}
001601      phraseFile:         TFileScanner;               {The Main Phrase File TFileScanner}
001602      process:            TProcess;                   {The process object of this process}
001603
001604      screenRightEdge:    INTEGER;                    {720 for Lisa 1.0 screen}
001605      scrollRgn:          RgnHandle;                  {what needs to be refreshed because of scroll}
001606      stdMargins:         LRect;                      {standard page-margins, in screen pixels}
001607      suspendSuffix:      ARRAY [1..maxSegments] OF STRING[3];
001608      theBodyPad:         TBodyPad;                   {current BodyPad being written to}
001609      theMarginPad:       TMarginPad;                 {current MarginPad being written to}
001610      toolName:           STRING[67];                 {The name of the tool}
001611      toolPrefix:         TFilePath;                  (*The prefix '{Tnn}' of the OS path name of the tool*)
001612      toolVolume:         TFilePath;                  {The volume '-name-' on which the tool resides}
001613      varPage:            STRING[20]; {+SW+}          {The string 'PAGE', for use in heading variables}
001614      varTitle:           STRING[20]; {+SW+}          {The string 'TITLE' for use in heading variables}
001615      wordDelimiters:     STRING[67];                 {The delimiters of a Lisa "word" in this language}
001616
001617
001618  PROCEDURE GetPrefixPart(wholeName: S255; VAR filePart: TFilePath); (*'{prefix}'*)
001619  FUNCTION  ToolOfFile(wholeName: S255): LONGINT;
001620  FUNCTION  ToolOfProcess(processId: LONGINT): LONGINT;
001621
001622  { Used to insert comments into the Universal Graph of Clipboard, so LisaDraw can understand it;
001623      These procedures only insert comment when we are generating the Universal Graph }
001624                                  { beginning of a series of text drawing ops that should be grouped }
001625  PROCEDURE PicTextBegin(alignment: TAlignment);
001626  PROCEDURE PicTextEnd;           { end of series }
001627  PROCEDURE PicGrpBegin;          { beginning of a series of grouped objects }
```

```
001628  PROCEDURE PicGrpEnd;            { end of series }
001629
001630  PROCEDURE InitProcess;
001631
001632  FUNCTION GetTime: LONGINT;
001633      {This function returns the same "time" as is used in events (see global variable eventTime),
001634              and in the idle loop}
001635
001636
001637  IMPLEMENTATION
001638
001639  {$I LIBTK/UABC2.TEXT}     {TProcess-TDocDirectory-TDocManager-TClipboard-TCommand-TCutCopyCommand-
001640                             TPasteCommand}
001641  {$I LIBTK/UABC3.TEXT}     {TImage-TView-TPaginatedView-TPageView-TPrintManager-THeading-TSelection}
001642  {$I LIBTK/UABC4.TEXT}     {TWindow-TDialogBox-TMenuBar-TFont}
001643  {$I LIBTK/UABC5.TEXT}     {TPanel-TBand-TPane-TMarginPad-TBodyPad-TScroller-TScrollBar}
001644
001645  (**********
001646  {$I UABC2.TEXT}     {TProcess-TDocDirectory-TDocManager-TClipboard-TCommand-TCutCopyCommand-TPasteCommand}
001647  {$I UABC3.TEXT}     {TImage-TView-TPaginatedView-TPageView-TPrintManager-THeading-TSelection}
001648  {$I UABC4.TEXT}     {TWindow-TDialogBox-TMenuBar-TFont}
001649  {$I UABC5.TEXT}     {TPanel-TBand-TPane-TMarginPad-TBodyPad-TScroller-TScrollBar}
001650  **********)
001651
001652  END.
001653

End of File -- Lines: 1653 Characters: 71219
```

```
================================================================================================
FILE: "LIBTK/UABC2.TEXT"
================================================================================================

000001  {INCLUDE FILE UABC2 -- IMPLEMENTATION OF UABC}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004          {TProcess-TDocDirectory-TDocManager-TClipboard-TCommand-TCutCopyCommand-TPasteCommand}
000005
000006
000007  {Segments: SgABCini(tialize and Terminate), SgABCres(ident), SgABCc(o)ld, SgABCdbg, SgABCpri(nting)}
000008
000009  {$IFC fRngABC}
000010  {$R+}
000011  {$ELSEC}
000012  {$R-}
000013  {$ENDC}
000014
000015  {$IFC fSymABC}
000016  {$D+}
000017  {$ELSEC}
000018  {$D-}
000019  {$ENDC}
000020
000021  CONST toolKitType = 9;
000022
000023      { picture comment IDs for pasting into LisaDraw }
000024        cPicGeDwg     =   100;
000025        cPicTxtBegin  =   101;
000026        cPicTxtEnd    =   102;
000027        cPicGrpBegin  =   103;
000028        cPicGrpEnd    =   104;
000029
000030  TYPE TPrPrfAlias = RECORD                                    {Alias for Print Preference}
000031                       CASE INTEGER OF
000032  {$IFC libraryVersion <= 20}        { P E P S I }
000033                           1: (prPrf: TPrPrf; prIns: TPrIns);
000034  {$ELSEC}
000035                           1: (prPrf: TPrRec);
000036  {$ENDC}                                 { S P R I N G}
000037                           2: (reserve: TPrReserve);
000038                       END;
000039
000040       TMapTable = RECORD                                      {Alias for menuBar.mapping TArray}
000041                       header:         TArrayHeader;
000042                       table:          ARRAY [1..8000] OF TWmgrCmd;
000043                   END;
```

```
000044       TMapPtr = ^TMapTable;
000045       TMapHandle = ^TMapPtr;
000046
000047  VAR
000048      alerts:             TAlertFile;                {The Alert Manager alert handle for the Main Phrase File}
000049      event:              EventRecord;               {The last event received by this process}
000050  {$IFC fDbgABC}
000051      hadToBindClip:      BOOLEAN;                   {BindHeap had to bind the Clipboard}
000052  {$ENDC}
000053      scrRgn1ForDrawHdgs: RgnHandle;                {Reserved for use dy TPaginatedView.AdornPageOnScreen}
000054      scrRgn2ForDrawHdgs: RgnHandle;                 {Reserved for use dy TPaginatedView.AdornPageOnScreen}
000055      wmgrMenus:          ARRAY [1..maxMenus] OF MenuInfo;
000056      cSelection:         TClass;                    {The TClass of TSelection, used by TPasteCmd.Perform}
000057      picData:            TH;                        {Pre-allocated handle on MainHeap used for picture
000058                                                             comments}
000059
000060
000061  PROCEDURE InAllMenusDo(iffLoaded: BOOLEAN; theCommand: TCmdNumber;
000062                         PROCEDURE doProc(VAR menu: MenuInfo; itemIndex: INTEGER)); FORWARD;
000063
000064
000065  {$S sScroll}
000066  PROCEDURE PreSbList(VAR sbList: TSbList; scrollBar: TScrollBar);
000067  BEGIN
000068      {$IFC fTrace}BP(1);{$ENDC}
000069      sbList.hz := POINTER(ORD(scrollBar.Heap));
000070      IF scrollBar.firstBox = NIL THEN
000071          sbList.hsbFst := hsbNil
000072      ELSE
000073          sbList.hsbFst := POINTER(scrollBar.firstBox.sBoxID);
000074      {$IFC fTrace}EP;{$ENDC}
000075  END;
000076
000077
000078  {$S sScroll}
000079  PROCEDURE PostSbList(sbList: TSbList; scrollBar: TScrollBar);
000080      VAR scroller: TScroller;
000081  BEGIN
000082      {$IFC fTrace}BP(1);{$ENDC}
000083      IF sbList.hsbFst = hsbNil THEN
000084          scroller := NIL
000085      ELSE
000086          scroller := POINTER(RefconSb(sbList.hsbFst));
000087      scrollBar.firstBox := scroller;
000088      {$IFC fTrace}EP;{$ENDC}
000089  END;
000090
000091
```

```
000092  {$S sStartup}
000093  PROCEDURE GetPrefixPart{(wholeName: S255; VAR filePart: TFilePath)}; (*'{prefix}'*)
000094          (* This works ONLY on Desktop Manager file names of the form '-volname-{prefix}suffix' *)
000095      VAR centerHyphen: INTEGER;
000096  BEGIN
000097      {$IFC fTrace}BP(1);{$ENDC}
000098      centerHyphen := Pos('-{', wholeName);
000099      filePart := Copy(wholeName, centerHyphen+1, Pos('}',wholeName) - centerHyphen);
000100      {$IFC fTrace}EP;{$ENDC}
000101  END;
000102
000103
000104  {$S sCldInit}
000105  FUNCTION  ToolOfFile{(wholeName: S255): LONGINT};
000106      VAR toolNumber: LONGINT;
000107          toolPrefix: TFilePath;
000108          cvResult:   TConvResult;
000109  BEGIN
000110      {$IFC fTrace}BP(7);{$ENDC}
000111      GetPrefixPart(wholeName, toolPrefix);
000112      Delete(toolPrefix, 1, 2); (* The '{T' *)
000113      Delete(toolPrefix, Length(toolPrefix), 1); (* The final '}' *)
000114      StrToLInt(@toolPrefix, toolNumber, cvResult);
000115      IF cvResult <> cvValid THEN
000116          ToolOfFile := 0
000117      ELSE
000118          ToolOfFile := toolNumber;
000119      {$IFC fTrace}EP;{$ENDC}
000120  END;
000121
000122
000123  {$S sCldInit}
000124  FUNCTION  ToolOfProcess{(processId: LONGINT): LONGINT};
000125      VAR prcsInfo:   ProcInfoRec;
000126          error:      INTEGER;
000127  BEGIN
000128      {$IFC fTrace}BP(6);{$ENDC}
000129      Info_Process(error, processID, prcsInfo);
000130      IF error > 0 THEN
000131          ToolOfProcess := 0
000132      ELSE
000133          ToolOfProcess := ToolOfFile(prcsInfo.progPathname);
000134      {$IFC fTrace}EP;{$ENDC}
000135  END;
000136
000137
000138  {$IFC fDbgABC}
000139  {$S SgABCdbg}
```

```
000140  PROCEDURE ReportEvent;
000141      VAR winTitle:       Str255;
000142  BEGIN
000143      Write(toolName, ' P=#', myProcessId:1, ' received ');
000144      WITH event DO
000145          BEGIN
000146          CASE what OF
000147              buttonDown:         Write('Button-down');
000148              buttonUp:           Write('Button-up');
000149              folderActivate:     Write('Activate');
000150              folderDeactivate:   Write('Deactivate');
000151              folderMoved:        Write('Window-moved');
000152              folderUpdate:       Write('Update');
000153              keyDown:            Write('Key-press');
000154              filerEvent:         Write('Desktop');
000155              OTHERWISE           Write('Miscellaneous');
000156              END;
000157          Write(' event for the ');
000158          IF who = alertFolder THEN
000159              WriteLn('Alert Box')
000160          ELSE
000161          IF who = dialogFolder THEN
000162              WriteLn('Dialog Box')
000163          ELSE
000164          IF who = scrapFolder THEN
000165              WriteLn('Clipboard')
000166          ELSE
000167          IF who = menuFolder THEN
000168              WriteLn('Menu Bar')
000169          ELSE
000170              BEGIN
000171              GetFldrTitle(who, winTitle);
000172              WriteLn('window titled "', winTitle, '"');
000173              END;
000174          END;
000175  END;
000176
000177
000178  {$S SgABCdbg}
000179  PROCEDURE ReportFilerEvent(flrParams: FilerExt);
000180  BEGIN
000181      Write('   ');
000182
000183      WITH flrParams DO
000184          BEGIN
000185          CASE theFlrOp OF
000186              fcClose:        Write('Close         ');
000187              fcCopy:         Write('Copy          ');
```

```
000188              fcDfClose:      Write('Doc File Close');
000189              fcNone:         Write('Open Tool     ');
000190              fcPut:          Write('Put           ');
000191              fcResume:       Write('Open Doc      ');
000192              fcShred:        Write('Shred         ');
000193              fcSuspend:      Write('Suspend       ');
000194              fcTerminate:    Write('Terminate     ');
000195              OTHERWISE       Write('Unknown!!!    ');
000196              END;
000197          {$IFC LibraryVersion <= 20}
000198          WriteLn('  theErr=', theErr:1, '  theDF=', theDF:1);
000199          WriteLn('  thePrefix="', thePrefix, '"');
000200          {$ELSEC}
000201          WriteLn('  theErr=', theErr:1, '  theOffset=', theOffset:1, '  theDF=', theDF:1);
000202          WriteLn('  thePassword="', thePassword, '"');
000203          WriteLn('  thePrefix="', thePrefix, '"');
000204          WriteLn('  theResult="', theResult, '"');
000205          {$ENDC}
000206          END;
000207  END;
000208  {$ENDC}
000209
000210
000211  {$S sError}
000212  PROCEDURE AlErrProc;
000213  BEGIN
000214      StopAlert(alerts, 2);
000215      process.Complete(FALSE);
000216  END;
000217
000218
000219  {$S sCldInit}
000220  FUNCTION  ExpandHeap(heap: THeap; bytesNeeded: INTEGER): INTEGER;
000221      VAR alias:          RECORD CASE INTEGER OF 1: (address: TPPrelude); 2: (high, low: INTEGER) END;
000222          preludePtr:     TPPrelude;
000223          oldHeapSize:    LONGINT;
000224          newHeapSize:    LONGINT;
000225  BEGIN
000226      {$IFC fTrace}BP(1);{$ENDC}
000227      alias.address := POINTER(ORD(heap));
000228      alias.low := 0;
000229      preludePtr := alias.address;
000230
000231      {$IFC fDbgABC}
000232      IF boundDocument.dataSegment.preludePtr <> preludePtr THEN
000233          ABCBreak('boundDocument''s preludePtr <> preludePtr in ExpandHeap', ORD(heap));
000234      {$ENDC}
000235
```

```
000236      oldHeapSize := CbOfHz(POINTER(ORD(heap)));
000237
000238      boundDocument.ExpandMemory(bytesNeeded);
000239
000240      WITH boundDocument.dataSegment.preludePtr^ DO
000241          newHeapSize := docSize - preludeSize;
000242
000243      ExpandHeap := newHeapSize - oldHeapSize;
000244      {$IFC fTrace}EP;{$ENDC}
000245  END;
000246
000247
000248  {$S SgABCcld}
000249  PROCEDURE PicTextBegin{(alignment: TAlignment)};
000250      TYPE    TpByte = ^Byte;
000251              ThByte = ^TpByte;
000252
000253      VAR     FEalign:    Byte;
000254  BEGIN
000255      IF genClipPic THEN
000256          BEGIN
000257          FEalign := ORD(alignment) + 1;
000258          IF FEalign > 3 THEN
000259              FEalign := 1; {aLeft}
000260          ThByte(picData)^^ := FEalign; {currently, picData is always a handle to 1 byte}
000261  {$IFC LibraryVersion <= 20}
000262          PicComment(cPicTxtBegin, SIZEOF(FEalign), Handle(picData));
000263  {$ELSEC}
000264          PicComment(cPicTxtBegin, SIZEOF(FEalign), QDHandle(picData));
000265  {$ENDC}
000266          END;
000267  END;
000268
000269
000270  {$S SgABCcld}
000271  PROCEDURE PicTextEnd;           { end of series }
000272  BEGIN
000273      IF genClipPic THEN
000274          PicComment(cPicTxtEnd, 0, NIL);
000275  END;
000276
000277
000278  {$S SgABCcld}
000279  PROCEDURE PicGrpBegin;          { beginning of a series of grouped objects }
000280  BEGIN
000281      IF genClipPic THEN
000282          PicComment(cPicGrpBegin, 0, NIL);
000283  END;
```

```
000284
000285
000286  {$S SgABCcld}
000287  PROCEDURE PicGrpEnd;            { end of series }
000288  BEGIN
000289      IF genClipPic THEN
000290          PicComment(cPicGrpEnd, 0, NIL);
000291  END;
000292
000293
000294  {$S sError}
000295  FUNCTION  FilerReason(error: INTEGER): FReason;
000296  BEGIN
000297      {$IFC fTrace}BP(1);{$ENDC}
000298      FilerReason := allOk;
000299      IF error > 0 THEN
000300          CASE error OF
000301               309, erNoDiskSpace:
000302                              FilerReason := noDiskSpace;
000303               306, 311, 315, erNoMemory:
000304                              FilerReason := noMemory;
000305             {$IFC LibraryVersion > 20}
000306              1294, erWrongPassword:
000307                              FilerReason := wrongPassword;
000308             {$ENDC}
000309              erBadData:      FilerReason := badData;
000310              erPassword, erVersion,
000311              955, 957, 958, erCantRead:
000312                              FilerReason := cantRead;
000313              961, 962, erCantWrite:
000314                              FilerReason := cantWrite;
000315              erDirtyDoc:     FilerReason := dirtyDoc;
000316              erNoMoreDocs:   FilerReason := noMoreDocs;
000317              erAborted:      FilerReason := aUserAbort;
000318              OTHERWISE       FilerReason := internalError;
000319              END;
000320      {$IFC fTrace}EP;{$ENDC}
000321  END;
000322
000323
000324  {$S SgABCini}
000325  PROCEDURE InitProcess;
000326      CONST
000327          maxNameLen  =   63;             {this definition must be consistent with the DeskTop Manager}
000328      TYPE
000329          TDeskLabel   =  RECORD          {this definition must be consistent with the DeskTop Manager}
000330                          version:    INTEGER;
000331                          name:       STRING[maxNameLen];
```

```
000332                      (** other stuff we are not interested in
000333                        kind:       INTEGER;
000334                        toolOnly:   BOOLEAN;
000335                        multiDoc:   BOOLEAN;
000336                        windLoc:    Rect;
000337                      {plus there are other fields added for Spring release}
000338                      **)
000339                        END;
000340          TPPathName  =   ^Pathname;
000341
000342      VAR copyright:  S255;
000343          prcsInfo:   ProcInfoRec;
000344          progName:   TFilePath;
000345          error:      INTEGER;
000346          toolLabel:  TDeskLabel;
000347          actual:     LONGINT;
000348          len:        INTEGER;
000349          pPathName:  TPPathname;
000350  BEGIN
000351      {$IFC fTrace}BP(1);{$ENDC}
000352     {Tool Kit Library copyright notice (application must have its own notice in addition)}
000353      copyright := 'Copyright 1983, 1984, Apple Computer, Inc.';
000354
000355     {Initialize Various Globals}
000356      idleTime := -1;
000357      inBackground := FALSE;
000358  {$IFC fDbgABC}
000359      fCountHeap := FALSE;
000360      fExperimenting := FALSE;
000361      eventDebug := FALSE;        {Don't trace window manager events}
000362  {$ENDC}
000363      activeWindowID := 0;
000364      allowAbort := TRUE;
000365      boundDocument := NIL;
000366      boundClipboard := NIL;
000367      closedDocument := NIL;
000368      currentWindow := NIL;
000369      currentDocument := NIL;
000370      docList := NIL;
000371      cursorShape := noCursor;
000372
000373     {Assign process and tool globals}
000374      myProcessID := my_id;
000375      Info_Process(error, myProcessID, prcsInfo);
000376      IF error > 0 THEN
000377          InitErrorAbort(error);
000378      progName := prcsInfo.progPathName;
000379      SplitFilePath(progName, toolVolume, toolPrefix);
```

```
000380      GetPrefixPart(progName, toolPrefix); (*'{Tnn}'*)
000381      myTool := ToolOfFile(progName);
000382
000383      {Read name of tool}
000384      pPathName := @progName;
000385      Read_Label(error, pPathName^, ORD(@toolLabel), SIZEOF(toolLabel), actual);
000386
000387      IF (error <= 0) AND (Length(toolLabel.name) > 0) AND (Length(toolLabel.name) <= maxNameLen) THEN
000388          toolName := toolLabel.name
000389      ELSE
000390          BEGIN
000391          LIntToStr(myTool, @toolName);
000392          toolName := CONCAT('Tool ', toolName);
000393          END;
000394      {$IFC fTrace}EP;{$ENDC}
000395  END;
000396
000397
000398  {$S SgABCcld} {Segmentation ???}
000399  FUNCTION GetTime: LONGINT;
000400  BEGIN
000401      {$IFC fTrace}BP(1);{$ENDC}
000402      GetTime := Time;
000403      {$IFC fTrace}EP;{$ENDC}
000404  END;
000405
000406
000407  METHODS OF TProcess;
000408
000409
000410      {$S SgABCini}
000411      FUNCTION  {TProcess.}CREATE{(object: TObject; heap: THeap): TProcess};
000412      BEGIN
000413          {$IFC fTrace}BP(7);{$ENDC}
000414          IF object = NIL THEN
000415              object := NewObject(heap, THISCLASS);
000416          SELF := TProcess(object);
000417          {$IFC fTrace}EP;{$ENDC}
000418      END;
000419
000420
000421      {$S sStartup}
000422      FUNCTION  {TProcess.}AbortRequest{: BOOLEAN};
000423      BEGIN
000424          {$IFC fTrace}BP(2);{$ENDC}
000425          IF allowAbort THEN
000426              AbortRequest := Abort {ask Window Manager}
000427          ELSE
```

```
000428              AbortRequest := FALSE;
000429          {$IFC fTrace}EP;{$ENDC}
000430      END;
000431
000432
000433      {$S SgABCcld}
000434
000435    { If allowAbort is FALSE, simply calls fs.XferSequential.
000436      Otherwise, transfers in increments of chunksize and sets fs.Error to erAborted IF command period
000437      is pressed during the transfer.  Returns with an incomplete transfer IF command period or any other
000438      error occurs during the transfer. }
000439      PROCEDURE {TProcess.}AbortXferSequential{(whichWay: xReadWrite; pFirst: Ptr;
000440                                                numBytes, chunksize: LONGINT; fs: TFileScanner)};
000441          VAR xferAmount:     LONGINT;
000442              actual:         LONGINT;
000443      BEGIN
000444          {$IFC fTrace}BP(6);{$ENDC}
000445          IF allowAbort THEN
000446              BEGIN
000447              actual := 0;
000448
000449              WHILE (numBytes > 0) AND (fs.error <= 0) AND
000450                      NOT (fs.atEnd  AND (whichWay = xRead)) DO
000451                  BEGIN
000452                  IF numbytes > chunksize THEN
000453                      xferAmount := chunksize
000454                  ELSE
000455                      xferAmount := numbytes;
000456
000457                  IF process.AbortRequest THEN
000458                      fs.error := erAborted
000459                  ELSE
000460                      BEGIN
000461                      fs.XferSequential(whichWay, pFirst, xferAmount);
000462                      xferAmount := fs.actual;
000463
000464                      {$IFC fDbgABC}
000465                      IF (xferAmount <= 0) AND (fs.error <= 0) THEN
000466                          ABCbreak('In TProcess.AbortXferSequential, fs.actual <= 0', xferAmount);
000467                      {$ENDC}
000468
000469                      actual := actual + xferAmount;
000470                      numbytes := numBytes - xferAmount;
000471                      pFirst := POINTER(ORD(pFirst) + xferAmount);
000472                      END;
000473                  END;
000474
000475              fs.actual := actual; {make believe we xferred it all at once}
```

```
000476              END
000477          ELSE
000478              fs.XferSequential(whichWay, pFirst, numBytes);
000479          {$IFC fTrace}EP;{$ENDC}
000480      END;
000481
000482
000483      {$S sAlert}
000484      PROCEDURE {TProcess.}ArgAlert{(whichArg: TArgAlert; argText: S255)};
000485      BEGIN
000486          {$IFC fTrace}BP(7);{$ENDC}
000487          ArgAlert(whichArg, argText);
000488          {$IFC fTrace}EP;{$ENDC}
000489      END;
000490
000491
000492      {$S sAlert}
000493      FUNCTION  {TProcess.}Ask{(phraseNumber: INTEGER): INTEGER};
000494      BEGIN
000495          {$IFC fTrace}BP(7);{$ENDC}
000496          ArgAlert(0, toolName);
000497          {$IFC LibraryVersion > 20}
000498          IF activeWindowID = 0 THEN
000499              Ask := BackgroundAlert(alerts, phraseNumber, AskProc)
000500          ELSE
000501          {$ENDC}
000502              Ask := AskAlert(alerts, phraseNumber);
000503          {$IFC fTrace}EP;{$ENDC}
000504      END;
000505
000506
000507      {$S sAlert}
000508      PROCEDURE {TProcess.}BeginWait{(phraseNumber: INTEGER)};
000509      BEGIN
000510          {$IFC fTrace}BP(7);{$ENDC}
000511          ArgAlert(0, toolName);
000512          WaitAlert(alerts, phraseNumber);
000513          {$IFC fTrace}EP;{$ENDC}
000514      END;
000515
000516
000517      {$S sStartup}
000518      PROCEDURE {TProcess.}BindCurrentDocument;
000519      BEGIN
000520          {$IFC fTrace}BP(6);{$ENDC}
000521          IF (boundDocument <> currentDocument) AND (boundDocument <> NIL) THEN
000522              boundDocument.Unbind;
000523
```

```
000524          IF (boundClipboard <> currentDocument) AND (boundClipboard <> NIL) THEN
000525              boundClipboard.Unbind;
000526
000527          IF currentDocument <> NIL THEN
000528              currentDocument.Bind;
000529          {$IFC fTrace}EP;{$ENDC}
000530      END;
000531
000532
000533      {$S sAlert}
000534      FUNCTION  {TProcess.}Caution{(phraseNumber: INTEGER): BOOLEAN};
000535      BEGIN
000536          {$IFC fTrace}BP(7);{$ENDC}
000537          ArgAlert(0, toolName);
000538          {$IFC LibraryVersion > 20}
000539          IF activeWindowID = 0 THEN
000540              Caution := (BackgroundAlert(alerts, phraseNumber, CautionProc) = ORD(TRUE))
000541          ELSE
000542          {$ENDC}
000543              Caution := CautionAlert(alerts, phraseNumber);
000544          {$IFC fTrace}EP;{$ENDC}
000545      END;
000546
000547
000548      {$S sStartup}
000549      PROCEDURE {TProcess.}ChangeCursor{(cursorNumber: TCursorNumber)};
000550      BEGIN
000551          {$IFC fTrace}BP(4);{$ENDC}
000552          IF cursorNumber <> cursorShape THEN
000553              BEGIN
000554              SELF.DoCursorChange(cursorNumber);
000555              cursorShape := cursorNumber;
000556              END;
000557
000558          IF cursorNumber > icrsLast THEN
000559              SetStdCursor(icrsEscape);
000560          {$IFC fTrace}EP;{$ENDC}
000561      END;
000562
000563
000564      {$S SgABCini}
000565      PROCEDURE {TProcess.}Commence{(phraseVersion: INTEGER)};
000566          VAR aFile:      TFile;
000567              cacheSize:  INTEGER;
000568              cacheBytes: INTEGER;
000569              i:          INTEGER;
000570              oneChar:    STRING[1];
000571              manualPat:  Pattern;
```

```
000572              error:      INTEGER;
000573              prPrfAlias: TPrPrfAlias;
000574              str:        S255;
000575              convResult: TConvResult;
000576      BEGIN
000577          {$IFC fTrace}BP(7);{$ENDC}
000578
000579          {Open Phrase File}
000580          aFile := TFile.CREATE(NIL, mainHeap, CONCAT(toolVolume, toolPrefix, 'PHRASE'), '');
000581          phraseFile := aFile.ScannerFrom(0, [fRead]);
000582          InitErrorAbort(phraseFile.error);
000583
000584          {Read Menus}
000585          menuBar := TMenuBar.CREATE(NIL, mainHeap, phraseFile);
000586
000587          {Initialize and Read Alerts}
000588          cacheSize := phraseFile.ReadNumber(2);
000589          cacheBytes := phraseFile.ReadNumber(2);
000590          InitErrorAbort(phraseFile.error);
000591          InitAlerts(cacheSize, cacheBytes, POINTER(ORD(mainHeap)), NIL, @AlErrProc);
000592          InitErrorAbort(alertError);
000593          alerts := ReadAlerts(phraseFile.refnum, phraseVersion);
000594          InitErrorAbort(alertError);
000595
000596          {Read Word Delimiters}
000597          GetAlert(alerts, phWordDelimiters, @wordDelimiters);
000598          IF Length(wordDelimiters) > 67 THEN
000599              BEGIN
000600              ABCBreak('More than 67 characters in the word delimiter string--phrase number', phWordDelimiters);
000601                { Set error to something so we don't continue }
000602              InitErrorAbort(erInternal);
000603              END;
000604
000605          {Read "OK" and "Cancel"}
000606          GetButn(0, @cancelString);
000607          StrUpperCased(@cancelString);
000608          GetButn(1, @okString);
000609          StrUpperCased(@okString);
000610
000611          GetAlert(alerts, phNewHeading, @dfltNewHeading); {+SW+}
000612          GetAlert(alerts, phPage, @varPage); {+SW+}
000613          GetAlert(alerts, phTitle, @varTitle); {+SW+}
000614
000615          GetAlert(alerts, phCountry, @str);
000616          StrToInt(@str, countryCode, convResult);
000617          IF convResult <> cvValid THEN
000618              countryCode := 0;
000619
```

```
000620          {Create a handle to use in picture comments}
000621          picData := HAllocate(THz(mainHeap), 1);
000622
000623  (****
000624          {Read Tool Name}
000625           GetAlert(alerts, phToolName, @toolName);
000626           IF Length(toolName) > 67 THEN
000627               BEGIN
000628               ABCBreak('More than 67 characters in the tool name string--phrase number', phToolName);
000629               InitErrorAbort(erInternal);
000630               END;
000631  ****)
000632          {Read Tool Name from file label is done in InitProcess}
000633
000634          IF onDesktop THEN
000635              BEGIN
000636             {Initialize Print Manager, while Alert Segment is still Resident}
000637  {$IFC LibraryVersion <= 20}
000638              PrMgrInit(error);
000639              InitErrorAbort(error);
000640  {$ELSEC}
000641              PrMgrInit;
000642  {$ENDC}
000643              END;
000644
000645         {Initialize Scroll Bar and Cursor Library}
000646          InitWmlSb;
000647          InitWmlCrs(error);
000648          InitErrorAbort(error);
000649
000650  {$IFC LibraryVersion <= 20 AND FALSE} {do it this way in case we need it back for the Pepsi version}
000651         {Create fonts}
000652          fonts[ 0] := TFont.CREATE(NIL, mainHeap, sysText);   {System Font     }
000653          fonts[ 1] := TFont.CREATE(NIL, mainHeap, p15Tile);   {15 pitch Gothic }
000654          fonts[ 2] := TFont.CREATE(NIL, mainHeap, p12tile);   {12 pitch Modern }
000655          fonts[ 3] := TFont.CREATE(NIL, mainHeap, elite);     {12 pitch Elite  }
000656          fonts[ 4] := TFont.CREATE(NIL, mainHeap, p10tile);   {10 pitch Modern }
000657          fonts[ 5] := TFont.CREATE(NIL, mainHeap, p10cent);   {10 pitch Courier}
000658          fonts[ 6] := TFont.CREATE(NIL, mainHeap, tile12 );   {PS Modern       }
000659          fonts[ 7] := TFont.CREATE(NIL, mainHeap, cent12 );   {PS Executive    }
000660          fonts[ 8] := TFont.CREATE(NIL, mainHeap, tile18 );   {1/4 inch Modern }
000661          fonts[ 9] := TFont.CREATE(NIL, mainHeap, cent18 );   {1/4 inch Classic}
000662          fonts[10] := TFont.CREATE(NIL, mainHeap, tile24 );   {1/3 inch Modern }
000663          fonts[11] := TFont.CREATE(NIL, mainHeap, cent24 );   {1/3 inch Classic}
000664  {$ENDC}
000665
000666         {Specify suspend-file suffixes}
000667          oneChar := '0';
```

```
000668          FOR i := 1 TO maxSegments DO
000669              BEGIN
000670              oneChar[1] := CHR(48+i);
000671              suspendSuffix[i] := CONCAT('$S', oneChar);
000672              END;
000673
000674         {Initialize other globals}
000675
000676          SetPt(zeroPt, 0, 0);
000677          SetRect(zeroRect, 0, 0, 0, 0);
000678          SetRect(hugeRect, 0, 0, $3FFF, $3FFF);
000679
000680          SetLPt(zeroLPt, 0, 0);
000681          SetLRect(zeroLRect, 0, 0, 0, 0);
000682          SetLRect(hugeLRect, 0, 0, $3FFFFFFF, $3FFFFFFF);
000683
000684          orthogonal[v] := h;
000685          orthogonal[h] := v;
000686
000687          docList := TList.CREATE(NIL, mainHeap, 1);
000688
000689          highToggle[FALSE] := hOnToOff;
000690          highToggle[TRUE]  := hOffToOn;
000691
000692          highLevel[FALSE] := hOffToDim;
000693          highLevel[TRUE]  := hOffToOn;
000694
000695          PenNormal;
000696          GetPenState(normalPen);
000697
000698          PenSize(2, 2);
000699
000700          PenMode(patXor);
000701          PenPat(gray);
000702          GetPenState(highPen[hDimToOff]);
000703          GetPenState(highPen[hOffToDim]);
000704
000705          PenMode(notPatXor);
000706          PenPat(gray);
000707          GetPenState(highPen[hOnToDim]);
000708          GetPenState(highPen[hDimToOn]);
000709
000710          PenMode(patXor);
000711          PenPat(black);
000712          GetPenState(highPen[hOffToOn]);
000713          GetPenState(highPen[hOnToOff]);
000714
000715          PenSize(3, 2);
```

```
000716          PenMode(patXOr);
000717
000718          PenPat(gray);
000719          GetPenState(autoBreakPen);
000720
000721          StuffHex(@manualPat, 'CC663399CC663399');
000722          PenPat(manualPat);
000723          GetPenState(manualBreakPen);
000724
000725          StuffHex(@marginPattern, '8000000008000000');
000726
000727          PenNormal;
000728          PenPat(manualPat);
000729          GetPenState(limboPen);
000730
000731          SetPt(screenRes, 90, 60);  {Lisa 1.0 screen}    {better--get from phrase file}
000732          screenRightEdge := 720;  {redundant -- screenBits.bounds.right shd be the same}
000733
000734          SetLRect(stdMargins, screenRes.h, screenRes.v, - screenRes.h, -screenRes.v);
000735
000736          PenNormal;
000737
000738          noPad := TPad.CREATE(NIL, mainHeap, zeroRect, hugeLRect, screenRes, screenRes, NIL);
000739
000740  (***** Do this in TPad creation block, via coercion
000741          noPad.PatToLPat(white, lPatWhite);
000742          noPad.PatToLPat(black, lPatBlack);
000743          noPad.PatToLPat(gray, lPatGray);
000744          noPad.PatToLPat(ltGray, lPatLtGray);
000745          noPad.PatToLPat(dkGray, lPatDkGray);
000746  *****)
000747
000748          MakeTypeStyle(famClassic, size18Point, [], cornerNumberStyle);
000749
000750          theMarginPad := TMarginPad.CREATE(NIL, mainHeap);
000751          theBodyPad := theMarginPad.bodyPad;
000752          IF crashPad = NIL THEN
000753              crashPad := theMarginPad;
000754
000755          clipboard := TClipboard.CREATE(NIL, mainHeap);
000756
000757          padRgn := NewRgn;
000758          focusRgn := thePort^.visRgn;
000759          focusStkPtr := 0;
000760          focusArea := NIL;
000761          genClipPic := FALSE;
000762          amPrinting := FALSE;
000763          useAltVisRgn := FALSE;
```

```
000764
000765          altVisRgn := NewRgn;
000766          scrollRgn := NewRgn;
000767
000768          scrRgn1ForDrawHdgs := NewRgn;
000769          scrRgn2ForDrawHdgs := NewRgn;
000770
000771          blinkOnCentiSecs := caretOnTime;
000772          blinkOffCentiSecs := caretOffTime;
000773
000774          PrPrfDefault(prPrfAlias.prPrf);
000775          clipPrintPref := prPrfAlias.reserve;
000776
000777          { Final check for Abort in init }
000778          InitErrorAbort(0);
000779
000780          {$IFC fTrace}EP;{$ENDC}
000781      END;
000782
000783
000784      {$S SgABCini}
000785      PROCEDURE {TProcess.}Complete{(allIsWell: BOOLEAN)};
000786          VAR s:          TListScanner;
000787              document:   TDocManager;
000788      BEGIN
000789          {$IFC fTrace}BP(7);{$ENDC}
000790          IF NOT (allIsWell OR amDying) THEN
000791              BEGIN
000792              ImDying; {Do this first}
000793              IF (boundClipboard <> NIL) AND (scrapProcess = myProcess) THEN {*** Sufficient & necessary? ***}
000794                  BackOutOfScrap;
000795              amDying := TRUE;
000796              END;
000797
000798          {$IFC fDbgABC}
000799          IF NOT allIsWell THEN
000800              ABCBreak('Process.Complete(FALSE)', 0);
000801          {$ENDC}
000802
000803          IF docList <> NIL THEN
000804              BEGIN
000805              s := docList.Scanner;
000806              docList := NIL;
000807              WHILE s.Scan(document) DO
000808                  document.Complete(allIsWell);
000809              END;
000810          HALT;
000811          {$IFC fTrace}EP;{$ENDC}
```

```
000812      END;
000813
000814
000815      {$S SgABCcld}
000816      PROCEDURE {TProcess.}CopyExternalDoc(VAR error: INTEGER; externalName, volumePrefix: TFilePath);
000817      BEGIN
000818          {$IFC fTrace}BP(6);{$ENDC}
000819          {$IFC fDbgABC}
000820          ABCbreak('TProcess.CopyExternalDoc was not overridden', 0);
000821          {$ENDC}
000822          {$IFC fTrace}EP;{$ENDC}
000823      END;
000824
000825
000826      {$S sAlert}
000827      PROCEDURE {TProcess.}CountAlert{(whichCtr: TAlertCounter; counter: INTEGER)};
000828      BEGIN
000829          {$IFC fTrace}BP(7);{$ENDC}
000830          CountAlert(whichCtr, counter);
000831          {$IFC fTrace}EP;{$ENDC}
000832      END;
000833
000834
000835      {$S sStartup}
000836      PROCEDURE {TProcess.}DoCursorChange{(cursorNumber: TCursorNumber)};
000837      BEGIN
000838          {$IFC fTrace}BP(4);{$ENDC}
000839          SetStdCursor(cursorNumber);
000840          {$IFC fTrace}EP;{$ENDC}
000841      END;
000842
000843
000844      {$IFC fDebugMethods}
000845      {$S SgABCini}
000846      PROCEDURE {TProcess.}DontDebug;
000847      BEGIN
000848          {$IFC fTrace}BP(6);{$ENDC}
000849          fCheckIndices := FALSE;
000850  {$IFC fDbgABC}
000851          eventDebug := FALSE;
000852          fCountHeap := FALSE;
000853          fExperimenting := FALSE;
000854  {$ENDC}
000855          {$IFC fTrace}EP;{$ENDC}
000856      END;
000857      {$ENDC}
000858
000859
```

```
000860      {$S sAlert}
000861      PROCEDURE {TProcess.}DrawAlert(phraseNumber: INTEGER; marginLRect: LRect);
000862          VAR rectInWindow:   Rect;
000863      BEGIN
000864          {$IFC fTrace}BP(7);{$ENDC}
000865          ArgAlert(0, toolName);
000866          thePad.LRectToRect(marginLRect, rectInWindow);
000867          DrawAlert(alerts, phraseNumber, rectInWindow);
000868          {$IFC fTrace}EP;{$ENDC}
000869      END;
000870
000871
000872      {$IFC fDbgABC}
000873      {$S SgABCdbg}
000874      PROCEDURE {TProcess.}DumpGlobals;
000875          VAR str: S8;
000876          PROCEDURE AbortDumpVar(pVariable: Ptr; nameAndType: S255);
000877          BEGIN
000878              IF CheckKeyPress('Process global variable dump') THEN
000879                  BEGIN
000880                  WriteLn;
000881                  WriteLn;
000882                  Exit(DumpGlobals);
000883                  END;
000884              DumpVar(pVariable, nameAndType);
000885          END;
000886      BEGIN
000887          WriteLn;
000888          WriteLn('--- IMPORTANT GLOBAL VARIABLES OF THE PROCESS ---');
000889          WriteLn;
000890          AbortDumpVar(@activeWindowID, 'activeWindowID: Ptr');
000891          AbortDumpVar(@allowAbort, 'allowAbort: BOOLEAN');
000892          AbortDumpVar(@boundClipboard, 'boundClipboard: TClipboard');
000893          AbortDumpVar(@boundDocument, 'boundDocument: TDocManager');
000894          AbortDumpVar(@clickState, Concat('clickState: RECORD where: Point; when: LONGINT;',
000895                                  'clickCount: INTEGER; fShift: BOOLEAN; fOption: BOOLEAN; fApple: BOOLEAN END'));
000896          AbortDumpVar(@clipboard, 'clipboard: TClipboard');
000897          AbortDumpVar(@closedBySuspend, 'closedBySuspend: BOOLEAN');
000898          AbortDumpVar(@closedDocument, 'closedDocument: TDocManager');
000899          AbortDumpVar(@currentDocument, 'currentDocument: TDocManager');
000900          AbortDumpVar(@currentWindow, 'currentWindow: TWindow');
000901          AbortDumpVar(@cursorShape, 'cursorShape: INTEGER');
000902          AbortDumpVar(@deferUpdate, 'deferUpdate: BOOLEAN');
000903          AbortDumpVar(@docList, 'docList: TList');
000904          AbortDumpVar(@genClipPic, 'genClipPic: BOOLEAN');
000905          AbortDumpVar(@idleTime, 'idleTime: LONGINT');
000906          AbortDumpVar(@inBackground, 'inBackground: BOOLEAN');
000907          AbortDumpVar(@menuBar, 'menuBar: TMenuBar');
```

```
000908          AbortDumpVar(@myProcessID, 'myProcessID: LONGINT');
000909          AbortDumpVar(@myTool, 'myTool: LONGINT');
000910          AbortDumpVar(@process, 'process: TProcess');
000911          AbortDumpVar(@toolName, 'toolName: STRING[67]');
000912          AbortDumpVar(@toolPrefix, 'toolPrefix: STRING[255]');
000913          AbortDumpVar(@toolVolume, 'toolVolume: STRING[255]');
000914          WriteLn;
000915          WriteLn;
000916      END;
000917      {$ENDC}
000918
000919
000920      {$S sAlert}
000921      PROCEDURE {TProcess.}EndWait;
000922      BEGIN
000923          {$IFC fTrace}BP(7);{$ENDC}
000924          {$IFC LibraryVersion <= 20}
000925          HideFolder(alertFolder);
000926          {$ELSEC}
000927          EndWaitAlert;
000928          {$ENDC}
000929          {$IFC fTrace}EP;{$ENDC}
000930      END;
000931
000932
000933      {$S sAlert}
000934      PROCEDURE {TProcess.}GetAlert{(phraseNumber: INTEGER; VAR theText: S255)};
000935      BEGIN
000936          {$IFC fTrace}BP(7);{$ENDC}
000937          GetAlert(alerts, phraseNumber ,@theText);
000938          {$IFC fTrace}EP;{$ENDC}
000939      END;
000940
000941
000942      {$S Override}
000943      FUNCTION  {TProcess.}NewDocManager{(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager};
000944      BEGIN
000945          {$IFC fTrace}BP(7);{$ENDC}
000946          NewDocManager := TDocManager.CREATE(NIL, mainHeap, volumePrefix);
000947          {$IFC fTrace}EP;{$ENDC}
000948      END;
000949
000950
000951      {$S sAlert}
000952      PROCEDURE {TProcess.}Note{(phraseNumber: INTEGER)};
000953      {$IFC LibraryVersion > 20}
000954          VAR dummy:  INTEGER;
000955      {$ENDC}
```

```
000956      BEGIN
000957          {$IFC fTrace}BP(7);{$ENDC}
000958          ArgAlert(0, toolName);
000959          {$IFC LibraryVersion > 20}
000960          IF activeWindowID = 0 THEN
000961              dummy := BackgroundAlert(alerts, phraseNumber, NoteProc)
000962          ELSE
000963          {$ENDC}
000964              NoteAlert(alerts, phraseNumber);
000965          {$IFC fTrace}EP;{$ENDC}
000966      END;
000967
000968
000969      { NOTE: StopCondition is checked only when no events are available.
000970        NOTE: StopCondition should not assume that a document is bound.  If all the process' windows are
000971                  inactive, StopCondition will be called before the process is suspended (to give you
000972                  a chance to regain control), but all the process' documents will be unbound.  You can
000973                  check for this situation by testing currentDocument for NIL.}
000974
000975      {$S sStartup}
000976      PROCEDURE {TProcess.}ObeyEvents{(FUNCTION StopCondition: BOOLEAN)};
000977          LABEL 9;
000978
000979          VAR selection:  TSelection;
000980
000981          PROCEDURE StopTest;
000982          BEGIN
000983              IF StopCondition THEN
000984                  BEGIN
000985                  LetOthersRun;
000986                  GOTO 9;
000987                  END;
000988          END;
000989
000990          PROCEDURE GetAndObeyEvent;
000991              LABEL   1;
000992          BEGIN
000993              {$IFC fTrace}BP(1);{$ENDC}
000994              GetEvent(event);
000995
000996              {$IFC fDbgABC}
000997              IF fExperimenting and eventDebug THEN
000998                  WITH event.who^.portRect DO
000999                      BEGIN
001000                      WriteLn('GetAndObeyEvent (event.who):', ORD(event.who));
001001                      WriteLn(left, top, right, bottom);
001002                      WriteLn(event.where.h, event.where.v);
001003                      END;
```

```
001004              {$ENDC}
001005
001006              IF ImActive THEN
001007                  IF SELF.AbortRequest THEN
001008                      IF event.what IN [keyDown, buttonDown, buttonUp] THEN
001009                          GOTO 1;
001010              SELF.ObeyTheEvent;
001011  1:
001012              {$IFC fTrace}EP;{$ENDC}
001013          END;
001014
001015      BEGIN
001016          {$IFC fTrace}BP(7);{$ENDC}
001017         {Shouldn't tell Filer initFailed after this}
001018          isInitialized := TRUE;
001019
001020         {Main event loop}
001021
001022         {NOTE: currentWindow <> NIL implies  (1) process is active OR
001023                                              (2) process is running in the background and has a document }
001024
001025          REPEAT
001026              WHILE NOT (ImActive OR amDying OR (currentWindow <> NIL)) DO
001027                  BEGIN
001028                  IF NOT EventAvail THEN
001029                      StopTest;
001030                  GetAndObeyEvent; {may suspend me}
001031                  END;
001032
001033              WHILE (ImActive OR (currentWindow <> NIL)) AND NOT amDying DO
001034                  IF EventAvail THEN
001035                      GetAndObeyEvent
001036                  ELSE
001037                      BEGIN
001038                      StopTest;
001039
001040                      currentWindow.Update(TRUE);
001041                      IF currentWindow.dialogBox <> NIL THEN
001042                          currentWindow.dialogBox.Update(TRUE);
001043
001044                      IF NOT (amDying OR eventAvail) THEN
001045                          BEGIN
001046                          selection := currentWindow.selectWindow.selectPanel.selection;
001047                          idleTime := Time;
001048                          selection.IdleBegin(idleTime);
001049                          WHILE NOT (amDying OR eventAvail) DO
001050                              selection.IdleContinue(Time);
001051                          IF NOT amDying THEN
```

```
001052                          selection.IdleEnd(Time);
001053                      END;
001054                  END;
001055
001056          UNTIL amDying;
001057
001058      9:
001059      {$IFC fTrace}EP;{$ENDC}
001060      END;
001061
001062
001063      {$S sStartup}
001064      PROCEDURE {TProcess.}ObeyFilerEvent;
001065          LABEL 1;
001066          TYPE
001067  {$IFC LibraryVersion <= 20}
001068              TFileOpKind = (fopNone, fopSuspend, fopSaveVersion);
001069  {$ELSEC}
001070              TFileOpKind = (fopNone, fopSuspend, fopSaveVersion, fopCopyDoc);
001071  {$ENDC}
001072
001073          VAR reply:          FReply;
001074              badReply:       FReply;
001075              reason:         FReason;
001076              window:         TWindow;
001077              openAsTool:     BOOLEAN;
001078              document:       TDocManager;
001079              flrParams:      FilerExt;
001080              flrOp:          FilerOp;
001081              volumePrefix:   TFilePath;
001082              wasSuspended:   BOOLEAN;
001083              fileOpKind:     TFileOpKind;
001084              doSuspend:      BOOLEAN;
001085              doSave:         BOOLEAN;
001086              error:          INTEGER;
001087
001088          PROCEDURE CheckAbort(abortReason: INTEGER);
001089          VAR i:          INTEGER;
001090              dsPathname: PathName;
001091          BEGIN
001092              IF abortReason = 0 THEN
001093                  IF SELF.AbortRequest THEN
001094                      abortReason := erAborted
001095                  ELSE
001096                      Exit(CheckAbort);
001097
001098              {$IFC fDbgABC}
001099              IF abortReason <> erAborted THEN
```

```
001100                BEGIN
001101                WriteLn('--------------------');
001102                ReportFilerEvent(flrParams);
001103                ABCbreak('TProcess.ObeyFilerEvent got an error (event listed above)', abortReason);
001104                END;
001105            {$ENDC}
001106
001107            IF (flrOp = fcResume) OR (flrOp = fcNone) THEN
001108                BEGIN
001109                IF window <> NIL THEN
001110                    PopFocus;
001111                IF wasSuspended THEN
001112                  { Close but don't kill the datasegs }
001113                    BEGIN
001114                    FOR i := 1 TO maxSegments DO
001115                        IF document.dataSegment.refnum[i] >= 0 THEN
001116                            BEGIN
001117                            dsPathName := Concat(document.files.volumePrefix, suspendSuffix[i]);
001118                            Close_DataSeg(error, document.dataSegment.refnum[i]);
001119                            document.dataSegment.refnum[i] := -1;
001120                            END
001121                    END
001122                ELSE
001123                    BEGIN
001124                    { In case the BlankStationery method was called and opened any files }
001125                    document.CloseFiles;
001126                    { kill any data segments that were created }
001127                    document.KillSegments(1, maxSegments);
001128                    END;
001129                { delete from docList, IF there, and free regardless }
001130                docList.DelObject(document, TRUE);
001131                boundDocument := NIL;
001132                END;
001133
001134            TellFiler(error, badReply, FilerReason(abortReason), event.who);
001135            GOTO 1;
001136        END;
001137
001138    BEGIN
001139        {$IFC fTrace}BP(7);{$ENDC}
001140        wasSuspended := FALSE;
001141        GetAddParams(error, event, flrParams);
001142        IF error > 0 THEN
001143            ABCBreak('GetAddParams', error);
001144        flrOp := flrParams.theFlrOp;
001145        allowAbort := TRUE; {??? should we assume this ???}
001146
001147  {$IFC fDbgABC}
```

```
001148          IF eventDebug THEN
001149              ReportFilerEvent(flrParams);
001150  {$ENDC}
001151
001152          CASE flrOp OF
001153              fcNone, fcResume:
001154                  BEGIN
001155                { The assumption for aborting here is things will, where possible, be cleaned up along the way
001156                  by anyone detecting the abort.  Things that have already happened after the abort is
001157                  detected will be cleaned up in CheckAbort.  The process will of course continue after the
001158                  abort. }
001159
001160                  IF (inBackground) AND (doclist.size > 0) THEN
001161                      TellFiler(error, docClosd, noMoreDocs, event.who)   {No multiple doc's in background}
001162
001163                  ELSE
001164                      BEGIN
001165                          { Set badReply in case Abort is detected }
001166                      badReply := docClosd;
001167
001168                      TakeWindow(event.who);
001169
001170                      WITH flrParams DO
001171                          BEGIN
001172                          openAsTool := flrOp = fcNone;
001173
001174                          IF openAsTool THEN
001175                              thePrefix := CONCAT(toolVolume, toolPrefix);
001176
001177                          document := SELF.NewDocManager(thePrefix, openAsTool);
001178                          {$IFC LibraryVersion > 20}
001179                          document.files.password := thePassword;
001180                          {$ENDC}
001181                          END;
001182
001183                      IF document = NIL THEN {application refused the request}
001184                          TellFiler(error, docClosd, noMoreDocs, event.who)
001185
001186                      ELSE
001187                          BEGIN
001188                          document.openedAsTool := openAsTool;
001189
001190                          SetPort(event.who); {so things like InvalRect in BlankStationery will work}
001191
001192                            { Returns Abort as error = erAborted }
001193                          document.Open(error, ORD(event.who), wasSuspended);
001194                          window := NIL; {so CheckAbort will not PopFocus}
001195                          CheckAbort(error);
```

```
001196
001197                          PushFocus;
001198                          window := document.window;
001199                          window.Focus;
001200                          window.Resize(FALSE);
001201                          CheckAbort(0);
001202
001203                          InvalRect(window.innerRect);
001204                          window.Update(TRUE);
001205                          CheckAbort(0);
001206                          PopFocus;
001207
001208                          IF event.who = activeFolder THEN {already active so we don't get a folderActivate}
001209                              BEGIN
001210                              PushFocus;
001211                              window.Focus; {window.Activate assumes focused}
001212                              currentDocument := document; {this must be set before calling TWindow.Activate}
001213                              window.Activate;
001214                              PopFocus;
001215                              END
001216                          ELSE
001217                              window.StashPicture(hOffToDim);
001218                          END;
001219                      END;
001220                  END; {fcNone/fcResume case}
001221
001222                { The assumption for aborting here is things will be cleaned up along the way by anyone
001223                  detecting the abort.  The process will of course continue after the abort. }
001224              fcClose, fcSuspend, fcCopy, fcPut, fcShred:
001225                  BEGIN
001226  {$IFC LibraryVersion <= 20}
001227                  fileOpKind := fopNone;
001228                  document := POINTER(GetFldrRefCon(event.who));
001229                  document.Bind;
001230  {$ELSEC}
001231                  IF (flrOp = fcCopy) AND (Length(flrParams.theResult) > 0) THEN
001232                      BEGIN
001233                      fileOpKind := fopCopyDoc;
001234                      document := NIL;
001235                      END
001236                  ELSE
001237                      BEGIN
001238                      fileOpKind := fopNone;
001239                      document := POINTER(GetFldrRefCon(event.who));
001240                      document.Bind;
001241                      END;
001242
001243  {$ENDC}
```

```
001244
001245                  CASE flrOp OF
001246                      fcClose, fcSuspend, fcShred:
001247                          BEGIN
001248                          IF flrOp = fcClose THEN
001249                              BEGIN
001250                              IF document.window.changes <> 0 THEN
001251                                  fileOpKind := fopSaveVersion;
001252                              END
001253                          ELSE
001254                              fileOpKind := fopSuspend;
001255
001256                          volumePrefix := document.files.volumePrefix;
001257                          reply := docClosd;
001258                          badReply := docNotClosd;
001259                          END;
001260                      OTHERWISE {fcCopy, fcPut}
001261                          BEGIN
001262  {$IFC LibraryVersion <= 20}
001263                          fileOpKind := fopSaveVersion;
001264  {$ELSEC}
001265                          IF fileOpKind <> fopCopyDoc THEN
001266                              fileOpKind := fopSaveVersion;
001267  {$ENDC}
001268                          volumePrefix := flrParams.thePrefix;
001269                          reply := docXfered;
001270                          badReply := docNotXfered;
001271                          END;
001272                      END;
001273                  allowAbort :=  NOT doSuspend; {for now all ops can be aborted except fcSuspend and fcShred}
001274
001275                  CheckAbort(0);
001276
001277                  IF document <> NIL THEN
001278                      document.ConserveMemory(0, TRUE {GC});
001279                  CheckAbort(0);
001280
001281                  CASE fileOpKind OF
001282                      fopSuspend:
001283                          IF document.files.shouldSuspend THEN
001284                              document.Suspend(error);     {*** we ignore the volumePrefix !!! ***}
001285
001286                      fopSaveVersion:
001287                          IF document.files.shouldToolSave OR NOT document.openedAsTool THEN
001288                              document.SaveVersion(error, volumePrefix, FALSE);
001289
001290  {$IFC LibraryVersion > 20}
001291                      fopCopyDoc:
```

```
001292                          SELF.CopyExternalDoc(error, flrParams.theResult, volumePrefix);
001293  {$ENDC}
001294                      END;
001295
001296                    { You cannot abort after SaveVersion or Suspend unless the abort was detected within
001297                      SaveVersion or Suspend and indicated by their returned error being erAborted }
001298                  IF error > 0 THEN
001299                      IF flrOp = fcShred THEN
001300                          BEGIN {try to close all files}
001301                          document.CloseFiles;
001302                          document.KillSegments(1, maxSegments);
001303                          error := 0; {always give a good reply to the filer}
001304                          END
001305                      ELSE
001306                          CheckAbort(error);
001307
001308                  TellFiler(error, reply, FilerReason(error), event.who);
001309
001310                  IF flrOp <> fcCopy THEN
001311                      BEGIN
001312                      closedDocument := document;
001313                      closedBySuspend := doSuspend;
001314                      END;
001315
001316                  allowAbort := TRUE;
001317                  END;
001318
001319              fcDfClose:
001320                  BEGIN
001321                  badReply := dfNotClosed;
001322                  Close_Object(error, flrParams.theDf);
001323                  CheckAbort(error);
001324                  TellFiler(error, dfClosed, allOk, event.who);
001325                  END;
001326
001327              fcTerminate:
001328                  amDying := TRUE;
001329              END;
001330
001331      1:  {$IFC fTrace}EP;{$ENDC}
001332      END;
001333
001334
001335      {$S sStartup}
001336      PROCEDURE {TProcess.}ObeyTheEvent;
001337                  {NOTE: For the duration of the event, we are focused on the eventWindow}
001338          VAR eventDocument:  TDocManager;
001339              eventWindow:    TWindow;
```

```
001340              dialogBox:      TDialogBox;
001341              paused:         BOOLEAN;
001342              pkEvent:        EventRecord;
001343          {$IFC fCheckHeap}
001344              numObjects:     INTEGER;
001345              docHeap:        THeap;
001346          {$ENDC}
001347
001348          FUNCTION  EvtWindow(VAR evt: EventRecord): TWindow;
001349          BEGIN
001350              {$IFC fTrace}BP(1);{$ENDC}
001351              EvtWindow := eventDocument.WindowWithId(ORD(evt.who));
001352              IF evt.what = keyDown THEN
001353                  BEGIN
001354                  dialogBox := currentWindow.dialogBox;
001355                  IF dialogBox <> NIL THEN
001356                      IF dialogBox.keyResponse = diDismissDialogBox THEN
001357                          dialogBox.BeDismissed
001358                      ELSE
001359            {+SW+}    IF (dialogBox.keyResponse = diAccept) AND (currentWindow.selectWindow = dialogBox) THEN
001360                          EvtWindow := dialogBox
001361                  END;
001362              {$IFC fTrace}EP;{$ENDC}
001363          END;
001364
001365      BEGIN
001366          {$IFC fTrace}BP(7);{$ENDC}
001367          eventTime := event.when;
001368          eventType := event.what;
001369
001370  {$IFC fDbgABC}
001371          IF eventDebug THEN
001372              ReportEvent;
001373  {$ENDC}
001374
001375          WITH event DO
001376
001377              IF what = buttonUp THEN
001378
001379              ELSE
001380              IF what = filerEvent THEN
001381                  SELF.ObeyFilerEvent
001382
001383              ELSE
001384              IF who <> alertFolder THEN
001385                  BEGIN
001386
001387                  IF what = folderActivate THEN
```

```
001388                  TakeControl(event, FALSE, FALSE);
001389
001390              eventDocument := currentDocument;
001391
001392              IF who = menuFolder THEN       {much changed}
001393                  BEGIN
001394                  eventWindow := currentWindow;
001395                  dialogBox := currentWindow.dialogBox;
001396                  IF dialogBox <> NIL THEN
001397                      IF dialogBox.menuResponse = diDismissDialogBox THEN
001398                          dialogBox.BeDismissed
001399                      ELSE
001400                      IF dialogBox.menuResponse = diAccept THEN
001401                          eventWindow := currentWindow.selectWindow;  {+SW+}
001402                  END
001403              ELSE
001404              IF who = dialogFolder THEN
001405                  eventWindow := currentWindow.dialogBox
001406              ELSE
001407              IF who = scrapFolder THEN
001408                  BEGIN
001409                  eventDocument := clipboard;
001410                  clipboard.Bind;
001411                  eventWindow := clipboard.window;
001412                  END
001413              ELSE IF who = NIL THEN {assuming that we cannot receive a private event directed
001414                                      towards a particular window}
001415                  BEGIN
001416                  eventWindow := NIL;
001417                  process.HandlePrivateEvent(what, fromProcess, when, userData);
001418                  END
001419              ELSE
001420                  BEGIN
001421                  eventDocument := POINTER(GetFldrRefCon(who));
001422                  IF eventDocument = NIL THEN
001423                      BEGIN
001424                      ABCBreak('GetFldrRefCon = NIL', ORD(who));
001425                      eventWindow := NIL;
001426                      END
001427                  ELSE
001428                      BEGIN
001429                      eventDocument.Bind;
001430                      eventWindow := EvtWindow(event);
001431                      END;
001432                  END;
001433
001434              IF eventWindow <> NIL THEN
001435                  BEGIN
```

```
001436                  PushFocus;
001437                  IF who = menuFolder THEN
001438                      eventWindow.Focus
001439                  ELSE
001440                      BEGIN
001441                      SetPort(event.who);
001442                      {$IFC fDbgABC}
001443                      IF fExperimenting and eventDebug THEN
001444                          WITH thePort^.portRect DO
001445                              BEGIN
001446                              WriteLn('Before LocalToGlobal (thePort):', ORD(thePort));
001447                              WriteLn(left, top, right, bottom);
001448                              WriteLn(where.h, where.v);
001449                              END;
001450                      {$ENDC}
001451
001452                      LocalToGlobal(where);
001453                      eventWindow.Focus;
001454
001455                      {$IFC fDbgABC}
001456                      IF fExperimenting and eventDebug THEN
001457                          WITH thePort^.portRect DO
001458                              BEGIN
001459                              WriteLn('Before GlobalToLocal (thePort):', ORD(thePort));
001460                              WriteLn(left, top, right, bottom);
001461                              WriteLn(where.h, where.v);
001462                              END;
001463                      {$ENDC}
001464
001465                      GlobalToLocal(where);
001466
001467                      {$IFC fDbgABC}
001468                      IF fExperimenting and eventDebug THEN
001469                          WITH thePort^.portRect DO
001470                              BEGIN
001471                              WriteLn('After GlobalToLocal (thePort):', ORD(thePort));
001472                              WriteLn(left, top, right, bottom);
001473                              WriteLn(where.h, where.v);
001474                              END;
001475                      {$ENDC}
001476                      END;
001477
001478                  IF deferUpdate THEN
001479                      IF (what <> keyDown) OR appleKey THEN
001480                          eventWindow.Update(TRUE);
001481
001482                  deferUpdate := FALSE;
001483
```

```
001484                  CASE what OF
001485                      abortEvent:
001486                          eventWindow.AbortEvent;
001487                      buttonDown:
001488                          IF who = menuFolder THEN
001489                              eventWindow.MenuEventAt(where)
001490                          ELSE
001491                              eventWindow.DownEventAt(where);
001492                      folderActivate:
001493                          BEGIN
001494                          currentDocument := eventDocument;
001495                          eventWindow.Activate;
001496                          END;
001497                      folderDeactivate:
001498                          eventWindow.Deactivate;
001499                      folderMoved:
001500                          BEGIN
001501                          eventWindow.Resize(TRUE);
001502                          process.RememberCommand(uMoveWindow);
001503                          END;
001504                      folderUpdate:
001505                          eventWindow.Update(TRUE);
001506                      keyDown:
001507                          IF eventWindow.selectPanel = NIL THEN
001508                              {$IFC fDbgABC} ABCBreak('ObeyTheEvent: selectPanel=NIL', 0) {$ENDC}
001509                          ELSE
001510                              REPEAT
001511                                  eventWindow.selectPanel.selection.DoKey(ascii,
001512                                                  keyCap, shiftKey, appleKey, codeKey);
001513
001514                                  IF PeekEvent(pkEvent) THEN
001515                                      paused := (ImActive AND SELF.AbortRequest) OR
001516                                                (eventWindow <> EvtWindow(pkEvent)) OR
001517                                                (pkEvent.what <> keyDown) OR
001518                                                ((pkEvent.what = keyDown) AND (pkEvent.AppleKey)) {LSR}
001519                                  ELSE
001520                                      paused := TRUE;
001521
001522                                  IF NOT paused THEN
001523                                      BEGIN
001524                                      GetEvent(event);
001525                                      eventTime := event.when;
001526                                      eventType := event.what;
001527                                      {$IFC fDbgABC}
001528                                      IF eventDebug THEN
001529                                          ReportEvent;
001530                                      {$ENDC}
001531                                      END
```

```
001532                                  ELSE
001533                                  IF eventWindow.selectPanel <> NIL THEN
001534                                      eventWindow.selectPanel.selection.KeyPause;
001535                                UNTIL paused;
001536                      END;
001537
001538                  IF (closedDocument = NIL) AND (currentWindow <> NIL) THEN
001539                      BEGIN {+SW+}
001540                      IF NOT deferUpdate THEN
001541                          BEGIN
001542                          IF currentWindow.dialogBox <> NIL THEN
001543                              currentWindow.dialogBox.Update(TRUE);
001544                          currentWindow.Update(TRUE);
001545                          END;
001546                      IF currentWindow.objectToFree <> NIL THEN {+SW+}
001547                          BEGIN
001548                          currentWindow.objectToFree.Free;
001549                          currentWindow.objectToFree := NIL
001550                          END;
001551                      END;
001552
001553                  PopFocus;
001554                  END;
001555              END;
001556
001557          IF closedDocument <> NIL THEN
001558              BEGIN
001559              closedDocument.Close(closedBySuspend);
001560              closedDocument.Free;
001561              closedDocument := NIL;
001562              END;
001563
001564          process.BindCurrentDocument; {This also unbinds the eventDocument, in the case where
001565                                        we got an event while inactive.}
001566
001567          {$IFC fCheckHeap AND fDbgABC}
001568          IF fCountHeap AND (event.what <> buttonUp) THEN
001569              BEGIN
001570              numObjects := CountHeap(mainHeap);
001571              Write('mainHeap has ', numObjects:1, ' objects');
001572              IF boundDocument <> NIL THEN
001573                  BEGIN
001574                  docHeap := boundDocument.docHeap;
001575                  IF docHeap <> NIL THEN
001576                      BEGIN
001577                      numObjects := CountHeap(docHeap);
001578                      Write('; boundDocument heap has ', numObjects:1, ' objects');
001579                      MarkHeap(docHeap, ORD(boundDocument.dataSegment.preludePtr^.docDirectory));
```

```
001580                  SweepHeap(docHeap, TRUE);
001581                  END;
001582              END;
001583          IF boundClipboard <> NIL THEN
001584              BEGIN
001585              docHeap := boundClipboard.docHeap;
001586              IF docHeap <> NIL THEN
001587                  BEGIN
001588                  numObjects := CountHeap(docHeap);
001589                  Write('; boundClipboard heap has ', numObjects:1, ' objects');
001590                  END;
001591              END;
001592          WriteLn;
001593          END;
001594      {$ENDC}
001595
001596      {$IFC fDebugMethods}
001597      IF docList.Size = 0 THEN
001598          SELF.DontDebug;
001599      {$ENDC}
001600      {$IFC fTrace}EP;{$ENDC}
001601  END;
001602
001603
001604  {$S sError}
001605  FUNCTION  {TProcess.}Phrase{(error: INTEGER)};
001606      VAR erStr: S255;
001607  BEGIN
001608      {$IFC fTrace}BP(5);{$ENDC}
001609      {client can override}
001610      {also, I should case on os error codes}
001611      CASE error OF
001612          erAborted : Phrase := phTerminated;
001613          OTHERWISE
001614              BEGIN
001615              {$IFC fTrace}
001616              (** SuErrText('OSERRS.ERR', error, @erStr); **)
001617              Writeln;
001618              Writeln('Error # ',  error, '; ', erStr);
001619              {$ENDC}
001620              Phrase := phUnknown;
001621              END;
001622          END;
001623      {$IFC fTrace}EP;{$ENDC}
001624  END;
001625
001626
001627  {$S SgABCcld}
```

```
001628      PROCEDURE {TProcess.}HandlePrivateEvent(typeOfEvent: INTEGER; fromProcess: LONGINT;
001629                                              when: LONGINT; otherData: LONGINT);
001630      BEGIN
001631          {$IFC fTrace}BP(7);{$ENDC}
001632          {$IFC fTrace}EP;{$ENDC}
001633      END;
001634
001635
001636      {$S sRes}
001637      PROCEDURE {TProcess.}RememberCommand{(cmdNumber: TCmdNumber)};
001638          LABEL   1;
001639
001640          PROCEDURE CallWouldAlert(VAR menu: MenuInfo; itemIndex: INTEGER);
001641          BEGIN
001642              WouldAlert(menu, itemIndex);
001643              GOTO 1;
001644          END;
001645
001646      BEGIN
001647          {$IFC fTrace}BP(5);{$ENDC}
001648          IF NOT menubar.GetCmdName(cmdNumber, NIL) THEN
001649              cmdNumber := uSomeCommand;
001650          InAllMenusDo(TRUE,  cmdNumber, CallWouldAlert);
001651          InAllMenusDo(FALSE, cmdNumber, CallWouldAlert);
001652      1:
001653          {$IFC fTrace}EP;{$ENDC}
001654      END;
001655
001656
001657      {$S sStartup}
001658      PROCEDURE {TProcess.}Run;
001659          FUNCTION UntilPowerOff: BOOLEAN;
001660          BEGIN
001661              UntilPowerOff := FALSE;
001662          END;
001663      BEGIN
001664          {$IFC fTrace}BP(7);{$ENDC}
001665          SELF.ObeyEvents(UntilPowerOff);
001666          {$IFC fTrace}EP;{$ENDC}
001667      END;
001668
001669
001670      {$S SgABCcld}
001671      PROCEDURE {TProcess.}SendEvent(typeOfEvent: INTEGER; targetProcess: LONGINT; otherData: LONGINT);
001672          VAR er: EventRecord;
001673      BEGIN
001674          {$IFC fTrace}BP(7);{$ENDC}
001675          IF typeOfEvent < firstPrivateEvent THEN
```

```
001676              BEGIN
001677              {$IFC fDbgABC}
001678              ABCbreak('Invalid event type passed to TProcess.SendEvent', typeOfEvent);
001679              {$ENDC}
001680              END
001681          ELSE
001682              BEGIN
001683              WITH er DO
001684                  BEGIN
001685                  who := NIL; {can't tell what window we are sending to}
001686                  what := typeOfEvent;
001687                  when := Time;
001688                  toProcess := targetProcess;
001689                  fromProcess := myProcessID;
001690                  userData := otherData;
001691                  END;
001692              SendEvent(er, targetProcess);
001693              END;
001694          {$IFC fTrace}EP;{$ENDC}
001695      END;
001696
001697
001698      {$S sAlert}
001699      PROCEDURE {TProcess.}Stop{(phraseNumber: INTEGER)};
001700      {$IFC LibraryVersion > 20}
001701          VAR dummy:  INTEGER;
001702      {$ENDC}
001703      BEGIN
001704          {$IFC fTrace}BP(7);{$ENDC}
001705          ArgAlert(0, toolName);
001706          {$IFC LibraryVersion > 20}
001707          IF activeWindowID = 0 THEN
001708              dummy := BackgroundAlert(alerts, phraseNumber, StopProc)
001709          ELSE
001710          {$ENDC}
001711              StopAlert(alerts, phraseNumber);
001712          {$IFC fTrace}EP;{$ENDC}
001713      END;
001714
001715      {$S sStartup}
001716      PROCEDURE {TProcess.}TrackCursor;       { assumes we are active; can't track the cursor if not }
001717          VAR cursorNumber:   TCursorNumber;
001718      BEGIN
001719          {$IFC fTrace}BP(3);{$ENDC}
001720          cursorNumber := noCursor;
001721          IF currentWindow.dialogBox <> NIL THEN
001722              BEGIN
001723              cursorNumber := currentWindow.dialogBox.CursorFeedback;
```

```
001724              IF cursorNumber = noCursor THEN
001725                  IF currentWindow.dialogBox.downInMainWindowResponse = diRefuse THEN        {was cantDown}
001726                      cursorNumber := arrowCursor;
001727              END;
001728          IF cursorNumber = noCursor THEN
001729              cursorNumber := currentWindow.CursorFeedback;
001730          IF cursorNumber = noCursor THEN
001731              cursorNumber := arrowCursor;
001732          SELF.ChangeCursor(cursorNumber);
001733          {$IFC fTrace}EP;{$ENDC}
001734      END;
001735
001736
001737  {$S SgABCini}
001738  BEGIN
001739      UnitAuthor('Apple');
001740      InitProcess;
001741  END;
001742
001743
001744  METHODS OF TDocDirectory;
001745
001746
001747      {$S SgABCini}
001748      FUNCTION  {TDocDirectory.}CREATE{(object: TObject; heap: THeap; itsWindow: TWindow;
001749                                                                  itsClassWorld: TClassWorld): TDocDirectory};
001750          VAR world:  TClassWorld;
001751      BEGIN
001752          {$IFC fTrace}BP(7);{$ENDC}
001753          IF object = NIL THEN
001754              object := NewObject(heap, THISCLASS);
001755          SELF := TDocDirectory(object);
001756          WITH world DO
001757              BEGIN
001758              infRecs := TArray(itsClassWorld.infRecs.Clone(heap));
001759              classes := TArray(itsClassWorld.classes.Clone(heap)); (*^*)
001760              authors := TArray(itsClassWorld.authors.Clone(heap)); (*^*)
001761              aliases := TArray(itsClassWorld.aliases.Clone(heap)); (*^*)
001762              END;
001763
001764          WITH SELF DO
001765              BEGIN
001766              window := itsWindow;
001767              classWorld := world;
001768              END;
001769          {$IFC fTrace}EP;{$ENDC}
001770      END;
001771
```

```
001772
001773      {$IFC fDebugMethods}
001774      {$S SgABCdbg}
001775      PROCEDURE {TDocDirectory.}Fields{(PROCEDURE Field(nameAndType: S255))};
001776      BEGIN
001777          Field('window: TWindow');
001778          Field('classList: TList');
001779      END;
001780      {$ENDC}
001781
001782
001783      {$S SgABCcld}
001784      PROCEDURE {TDocDirectory.}Adopt; (*^*)
001785          VAR world:  TClassWorld;
001786              heap:   THeap;
001787      BEGIN
001788          {$IFC fMaxTrace}BP(1);{$ENDC}
001789          {$IFC fMaxTrace}EP;{$ENDC}
001790          heap := SELF.Heap;
001791          world := SELF.classWorld;
001792          WITH world DO
001793              BEGIN
001794              infRecs.Free;
001795              classes.Free;
001796              authors.Free;
001797              aliases.Free;
001798              infRecs := TArray(myWorld.infRecs.Clone(heap));
001799              classes := TArray(myWorld.classes.Clone(heap));
001800              authors := TArray(myWorld.authors.Clone(heap));
001801              aliases := TArray(myWorld.aliases.Clone(heap));
001802              END;
001803          SELF.classWorld := world;
001804      END;
001805
001806
001807  {$S SgABCini}
001808  END;
001809
001810
001811  METHODS OF TDocManager;
001812
001813
001814      {$S SgABCini}
001815      FUNCTION  {TDocManager.}CREATE{(object: TObject; heap: THeap; itsPathPrefix: TFilePath): TDocManager};
001816          VAR itsVolume:  TFilePath;
001817              itsFile:    TFilePath;
001818              i:          INTEGER;
001819      BEGIN
```

```
001820          {$IFC fTrace}BP(7);{$ENDC}
001821          IF object = NIL THEN
001822              object := NewObject(heap, THISCLASS);
001823          SELF := TDocManager(object);
001824          SplitFilePath(itsPathPrefix, itsVolume, itsFile);
001825          WITH SELF.files DO
001826              BEGIN
001827              volumePrefix := itsPathPrefix;
001828              volume := itsVolume;
001829  {$IFC LibraryVersion > 20}
001830              password := '';
001831  {$ENDC}
001832              shouldSuspend := TRUE;
001833              shouldToolSave := FALSE;
001834              END;
001835          WITH SELF.dataSegment DO
001836              BEGIN
001837              preludePtr := NIL;
001838              FOR i := 1 TO maxSegments DO
001839                  refnum[i] := -1;
001840              changes := 0;
001841              END;
001842          WITH SELF DO
001843              BEGIN
001844              window := NIL;
001845              pendingNote := 0;
001846              docHeap := NIL;
001847              END;
001848          {$IFC fTrace}EP;{$ENDC}
001849      END;
001850      {$S SgABCres}
001851
001852
001853      {$IFC fDebugMethods}
001854      {$S SgABCdbg}
001855      PROCEDURE {TDocManager.}Fields{(PROCEDURE Field(nameAndType: S255))};
001856      BEGIN (* TFilePath = STRING[255]; maxSegments = 6 *)
001857          Field(CONCAT('files: RECORD volumePrefix: STRING[255]; volume: STRING[255]; password: STRING[32];',
001858                       'saveExists: BOOLEAN; shouldSuspend: BOOLEAN; shouldToolSave: BOOLEAN; END'));
001859          Field('dataSegment: RECORD refnum: ARRAY [1..6] OF INTEGER; preludePtr: Ptr; changes: LONGINT; END');
001860          Field('docHeap: Ptr');
001861          Field('window: TWindow');
001862          Field('pendingNote: INTEGER');
001863          Field('openedAsTool: BOOLEAN');
001864          Field('');
001865      END;
001866      {$S SgABCres}
001867      {$ENDC}
```

```
001868
001869
001870      {$S SgABCcld}
001871      PROCEDURE {TDocManager.}Assimilate{(VAR error: INTEGER)};
001872          VAR hz:             THz;
001873              exDocDirectory: TDocDirectory;
001874              exClasses:      TClassWorld;
001875              doConvert:      BOOLEAN;
001876              olderVersion:   BOOLEAN;
001877              newerVersion:   BOOLEAN;
001878      BEGIN
001879          {$IFC fTrace}BP(7);{$ENDC}
001880          hz := POINTER(ORD(SELF.docHeap));
001881          hz^.procCbMore := @ExpandHeap; {The code address may have changed}
001882
001883          error := 0;
001884          WITH SELF.dataSegment.preludePtr^ DO
001885              BEGIN
001886              exDocDirectory := docDirectory;
001887              exClasses := exDocDirectory.classWorld;
001888              IF password <> 25376 THEN  {***temporary***}
001889                  error := erPassword;
001890              END;
001891
001892  (**)    IF error <= 0 THEN
001893              IF NeedConversion(exClasses, olderVersion, newerVersion) THEN
001894                  BEGIN
001895                  IF newerVersion THEN
001896                      doConvert := process.Caution(phNewerVersion)
001897                  ELSE
001898                  IF olderVersion THEN
001899                      doConvert := process.Caution(phOlderVersion)
001900                  ELSE
001901                      doConvert := TRUE;
001902
001903                  IF doConvert THEN
001904                      BEGIN
001905                      process.BeginWait(phConverting);
001906                      allowAbort := FALSE; {cannot abort the conversion}
001907
001908                      ConvertHeap(SELF.docHeap, exClasses);
001909                      exDocDirectory.Adopt; (*^*)
001910                      SELF.ConserveMemory(docExcess, TRUE {GC});
001911
001912                      allowAbort := TRUE;
001913                      process.EndWait;
001914                      END
001915                  ELSE
```

```
001916                  error := erVersion;
001917              END;                                (**)
001918          {$IFC fTrace}EP;{$ENDC}
001919      END;
001920
001921
001922      {$S sStartup}
001923      PROCEDURE {TDocManager.}Bind;
001924          VAR i:          INTEGER;
001925              error:      INTEGER;
001926              sched_err:  INTEGER;
001927      BEGIN
001928          {$IFC fTrace}BP(7);{$ENDC}
001929          IF boundDocument <> SELF THEN
001930              BEGIN
001931              IF boundDocument <> NIL THEN
001932                  boundDocument.Unbind;
001933
001934              i := 1;     {We must bind segment #1 before we can find out numSegments}
001935              REPEAT
001936                  Sched_Class(sched_err, FALSE);
001937                  Bind_DataSeg(error, SELF.dataSegment.refnum[i]);
001938                  Sched_Class(sched_err, TRUE);
001939
001940                  IF error > 0 THEN
001941                      ABCBreak('Bind_DataSeg', error);
001942                  i := i + 1;
001943              UNTIL i > SELF.dataSegment.preludePtr^.numSegments;
001944
001945              boundDocument := SELF;
001946              END;
001947          {$IFC fTrace}EP;{$ENDC}
001948      END;
001949
001950
001951      {$S SgABCcld}
001952      PROCEDURE {TDocManager.}Close{(afterSuspend: BOOLEAN)};
001953      BEGIN
001954          {$IFC fTrace}BP(7);{$ENDC}
001955          IF SELF = currentDocument THEN
001956              BEGIN
001957              currentDocument := NIL;
001958              currentWindow := NIL;
001959              activeWindowID := 0;
001960              END;
001961
001962          IF NOT afterSuspend THEN
001963              SELF.KillSegments(1, maxSegments);
```

```
001964
001965          docList.DelObject(SELF, FALSE);
001966          IF SELF = boundDocument THEN
001967              boundDocument := NIL;
001968          {$IFC fTrace}EP;{$ENDC}
001969      END;
001970      {$S SgABCres}
001971
001972
001973      {$S SgABCcld}
001974      PROCEDURE {TDocManager.}CloseFiles;
001975      BEGIN
001976          {$IFC fTrace}BP(7);{$ENDC}
001977          { For the application to override IF it needs to close any of its own files }
001978          {$IFC fTrace}EP;{$ENDC}
001979      END;
001980      {$S SgABCres}
001981
001982
001983      {$S SgABCini}
001984      PROCEDURE {TDocManager.}Complete{(allIsWell: BOOLEAN)};
001985      BEGIN
001986          {$IFC fTrace}BP(7);{$ENDC}
001987          {**** Try to save the document, code needed here. ****}
001988          {$IFC fTrace}EP;{$ENDC}
001989      END;
001990      {$S SgABCres}
001991
001992
001993      {$S SgABCcld}
001994      PROCEDURE {TDocManager.}ConserveMemory{(maxExcess: LONGINT; fGC: BOOLEAN)};
001995          VAR heap:           THeap;
001996              hz:             THz;
001997              bytesReduced:   LONGINT;
001998              error:          INTEGER;
001999      BEGIN
002000          {$IFC fTrace}BP(7);{$ENDC}
002001          IF SELF <> clipboard THEN
002002              BEGIN
002003              heap := SELF.docHeap;
002004
002005              IF fGC THEN
002006                  BEGIN
002007                  MarkHeap(heap, ORD(SELF.dataSegment.preludePtr^.docDirectory));
002008                  {$IFC fDbgABC}
002009                  SweepHeap(heap, TRUE);      {Report garbage}
002010                  {$ELSEC}
002011                  SweepHeap(heap, FALSE);     {Free garbage}
```

```
002012                  {$ENDC}
002013                  END;
002014
002015              hz := POINTER(ORD(heap));
002016              REPEAT
002017                  bytesReduced := CbShrinkHz(hz, maxSegSize)
002018              UNTIL bytesReduced < maxSegSize;
002019
002020              SELF.SetSegSize(error, CbOfHz(hz) + SELF.dataSegment.preludePtr^.preludeSize, maxExcess);
002021
002022              IF error > 0 THEN
002023                  process.Complete(FALSE);
002024              END;
002025          {$IFC fTrace}EP;{$ENDC}
002026      END;
002027      {$S SgABCres}
002028
002029
002030      {$S SgABCcld}
002031      PROCEDURE {TDocManager.}Deactivate;
002032      BEGIN
002033          {$IFC fTrace}BP(7);{$ENDC}
002034          IF SELF = currentDocument THEN
002035              BEGIN
002036              currentWindow := NIL;
002037              currentDocument := NIL; {so we can unbind the document}
002038              END;
002039
002040          allowAbort := FALSE;
002041          SELF.ConserveMemory(docExcess, FALSE {no GC});
002042          allowAbort := TRUE;
002043
002044          SELF.Unbind;
002045          {$IFC fTrace}EP;{$ENDC}
002046      END;
002047      {$S SgABCres}
002048
002049
002050      {$S SgABCini}
002051      FUNCTION  {TDocManager.}DfltHeapSize{: LONGINT};
002052      BEGIN
002053          {$IFC fTrace}BP(3);{$ENDC}
002054          DfltHeapSize := docDsBytes;
002055          {$IFC fTrace}EP;{$ENDC}
002056      END;
002057      {$S SgABCres}
002058
002059
```

```
002060      {$IFC fDbgABC}
002061      {$S SgABCdbg}
002062      PROCEDURE {TDocManager.}DumpPrelude;
002063          VAR preludePtr: TPPrelude;  {needed so WITH doesn't complain about $H+}
002064
002065          PROCEDURE AbortDumpVar(pVariable: Ptr; nameAndType: S255);
002066          BEGIN
002067              IF CheckKeyPress('Document prelude dump') THEN
002068                  BEGIN
002069                  WriteLn;
002070                  WriteLn;
002071                  Exit(DumpPrelude);
002072                  END;
002073              DumpVar(pVariable, nameAndType);
002074          END;
002075      BEGIN
002076          WriteLn;
002077          WriteLn('--- PRELUDE OF THE DOCUMENT ---');
002078          WriteLn;
002079          preludePtr := SELF.dataSegment.preludePtr;
002080          WITH preludePtr^ DO
002081              BEGIN
002082              AbortDumpVar(@password, 'password: INTEGER');
002083              AbortDumpVar(@version, 'version: INTEGER');
002084              AbortDumpVar(@country, 'country: INTEGER');
002085              AbortDumpVar(@language, 'language: INTEGER');
002086              AbortDumpVar(@preludeSize, 'preludeSize: INTEGER');
002087              AbortDumpVar(@docSize, 'docSize: LONGINT');
002088              AbortDumpVar(@numSegments, 'numSegments: INTEGER');
002089              AbortDumpVar(@docDirectory, 'docDirectory: TDocDirectory');
002090              END;
002091          WriteLn;
002092          WriteLn;
002093      END;
002094      {$S SgABCres}
002095      {$ENDC}
002096
002097
002098      {$S sCldInit}
002099      PROCEDURE {TDocManager.}ExpandMemory{(bytesNeeded: LONGINT)};
002100          VAR error:              INTEGER;
002101      BEGIN
002102          {$IFC fTrace}BP(7);{$ENDC}
002103          SELF.SetSegSize(error, SELF.dataSegment.preludePtr^.docSize + bytesNeeded, docExcess);
002104          IF error > 0 THEN
002105              process.Complete(FALSE);
002106          {$IFC fTrace}EP;{$ENDC}
002107      END;
```

```
002108      {$S SgABCres}
002109
002110
002111      {$S SgABCcld}
002112      PROCEDURE {TDocManager.}KillSegments{(first, last: INTEGER)};
002113          VAR i:              INTEGER;
002114              dsPathname:     PathName;
002115              {$IFC LibraryVersion > 20}
002116              dsPassword:     E_Name;
002117              blankPasswd:    E_Name;
002118              {$ENDC}
002119              error:          INTEGER;
002120      BEGIN
002121          {$IFC fTrace}BP(7);{$ENDC}
002122          error := 0;
002123          {$IFC LibraryVersion > 20}
002124          dsPassword := SELF.files.password;
002125          blankPasswd := '';
002126          {$ENDC}
002127          FOR i := first TO last DO
002128              IF SELF.dataSegment.refnum[i] >= 0 THEN
002129                  BEGIN
002130                  dsPathName := CONCAT(SELF.files.volumePrefix, suspendSuffix[i]);
002131                  {$IFC LibraryVersion > 20}
002132                  Change_Password(error, dsPathname, dsPassword, blankPasswd);
002133                  {$ENDC}
002134                  Kill_DataSeg(error, dsPathname);
002135                  Close_DataSeg(error, SELF.dataSegment.refnum[i]);
002136                  SELF.dataSegment.refnum[i] := -1;
002137                  END;
002138          {$IFC fTrace}EP;{$ENDC}
002139      END;
002140      {$S SgABCres}
002141
002142
002143      {$S sCldInit}
002144      PROCEDURE {TDocManager.}MakeSegments{(VAR error: INTEGER; oldSegments: INTEGER; newDocSize: LONGINT)};
002145          TYPE    TempType    = ARRAY [1..MAXINT] OF Byte;
002146                  PTempType   = ^TempType;
002147          VAR currDocSize:    LONGINT;
002148              newSegments:    INTEGER;
002149              i:              INTEGER;
002150              ldsn:           INTEGER;
002151              thisSegSize:    LONGINT;
002152              dsPathname:     PathName;
002153              dsRefnum:       INTEGER;
002154              memOrd:         LONGINT;
002155              dsInfo:         DsInfoRec;
```

```
002156              newSize:        LONGINT;
002157              p:              PTempType;
002158              {$IFC LibraryVersion > 20}
002159              dsPassword:     E_Name;
002160              blankPasswd:    E_Name;
002161              {$ENDC}
002162              sched_err:      INTEGER;
002163      BEGIN
002164          {$IFC fTrace}BP(7);{$ENDC}
002165          IF (boundDocument <> NIL) AND ((boundDocument <> SELF) OR (oldSegments = 0)) THEN
002166              boundDocument.Unbind;   {*** This may be dispensable ***}
002167
002168          error := 0;
002169
002170          IF (oldSegments > 0) THEN
002171              BEGIN
002172              {expand the current last data segment out to maxSegSize;
002173                  we assume that the caller has already checked that a new segment is actually needed}
002174
002175              dsRefnum := SELF.dataSegment.refnum[oldSegments];
002176
002177              Info_DataSeg(error, dsRefnum, dsInfo);
002178
002179              IF error <= 0 THEN
002180                  BEGIN
002181                  Sched_Class(sched_err, FALSE);
002182                  Size_DataSeg(error, dsRefnum, maxSegSize - dsInfo.mem_size,  newSize,
002183                                                maxSegSize - dsInfo.disc_size, newSize);
002184                  Sched_Class(sched_err, TRUE);
002185                  END
002186              ELSE
002187                  ABCbreak('In MakeSegments, error from Info_Dataseg', error);
002188              END;
002189
002190          currDocSize := oldSegments*maxSegSize;
002191          newSegments := oldSegments;
002192
002193          {$IFC LibraryVersion > 20}
002194          dsPassword := SELF.files.password;
002195          blankPasswd := '';
002196          {$ENDC}
002197
002198          WHILE (currDocSize < newDocSize) AND (error <= 0) DO
002199              BEGIN
002200              newSegments := newSegments + 1;
002201              ldsn := newSegments + docLdsn-1;
002202              thisSegSize := Min(newDocSize - currDocSize, maxSegSize);
002203              thisSegSize := LIntMulInt(LIntDivInt(thisSegSize + 511, 512), 512);
```

```
002204              dsPathname := CONCAT(SELF.files.volumePrefix, suspendSuffix[newSegments]);
002205
002206              {$IFC LibraryVersion > 20}
002207              Change_Password(error, dsPathname, dsPassword, blankPasswd);
002208              {$ENDC}
002209
002210              Open_Dataseg(error, dsPathname, dsRefnum, memOrd, ldsn);
002211
002212              {$IFC fDbgABC}
002213              IF error > 0 THEN
002214                  WriteLn('In TDocManager.MakeSegments: error from Open_Dataseg=', error:1);
002215              {$ENDC}
002216
002217              IF error > 0 THEN
002218                  BEGIN
002219                  Sched_Class(sched_err, FALSE);
002220                  Make_Dataseg(error, dsPathname, thisSegSize, thisSegSize, dsRefnum, memOrd, ldsn, ds_shared);
002221                  Sched_Class(sched_err, TRUE);
002222                  END
002223              ELSE
002224                  BEGIN
002225                  SetAccess_DataSeg(error, dsRefnum, FALSE);  {Make writeable}
002226                  IF error <= 0 THEN
002227                      BEGIN
002228                      Info_DataSeg(error, dsRefnum, dsInfo);
002229                      IF error <= 0 THEN
002230                          BEGIN
002231                          Sched_Class(sched_err, FALSE);
002232                          Size_DataSeg(error, dsRefnum, thisSegSize - dsInfo.mem_size, newSize,
002233                                                        thisSegSize - dsInfo.disc_size, newSize);
002234                          Sched_Class(sched_err, TRUE);
002235                          END;
002236                      END;
002237                  END;
002238
002239              IF error > 0 THEN
002240                  ABCBreak('In TDocManager.MakeSegments: Make_Dataseg', error)
002241              ELSE
002242                  BEGIN
002243                  {$IFC LibraryVersion > 20}
002244                  Change_Password(error, dsPathname, blankPasswd, dsPassword);
002245                  IF error > 0 THEN
002246                      ABCBreak('In TDocManager.MakeSegments: Change_Password', error);
002247                  {$ENDC}
002248
002249                  SELF.dataSegment.refnum[newSegments] := dsRefnum;
002250                  IF ldsn = docLdsn THEN
002251                      p := POINTER(memOrd);
```

```
002252                  END;
002253
002254              currDocSize := currDocSize + thisSegSize;
002255              IF process.AbortRequest THEN
002256                  error := erAborted;
002257              END;
002258
002259          IF error <= 0 THEN
002260              WITH SELF.dataSegment DO
002261                  BEGIN
002262                  IF oldSegments = 0 THEN
002263                      BEGIN
002264                      boundDocument := SELF;
002265                      FOR i := 1 TO SIZEOF(TPrelude) DO
002266                          p^[i] := 0;
002267                      preludePtr := POINTER(ORD(p));
002268                      END;
002269                  preludePtr^.docSize := currDocSize;
002270                  preludePtr^.numSegments := newSegments;
002271                  END;
002272          {$IFC fTrace}EP;{$ENDC}
002273      END;
002274
002275
002276      {$S Override}
002277      FUNCTION  {TDocManager.}NewWindow{(heap: THeap; wmgrID: TWindowID): TWindow};
002278      BEGIN
002279          {$IFC fTrace}BP(7);{$ENDC}
002280          NewWindow := TWindow.CREATE(NIL, heap, wmgrID, TRUE);
002281          {$IFC fTrace}EP;{$ENDC}
002282      END;
002283
002284
002285      {$S sCldInit}
002286      PROCEDURE {TDocManager.}Open{(VAR error: INTEGER; wmgrID: TWindowID; VAR openedSuspended: BOOLEAN)};
002287          LABEL   1;
002288
002289          VAR aFile:          TFile;
002290              volumePrefix:   TFilePath;
002291              pWindow:        WindowPtr;
002292              window:         TWindow;
002293      BEGIN
002294          {$IFC fTrace}BP(7);{$ENDC}
002295          openedSuspended := FALSE;
002296          volumePrefix := SELF.files.volumePrefix;
002297
002298          IF SELF.files.shouldToolSave OR NOT SELF.openedAsTool THEN
002299              BEGIN
```

```
002300              aFile := TFile.CREATE(NIL, mainHeap, volumePrefix, '');
002301
002302              {Look for the save file}
002303              IF NOT aFile.Exists(error) THEN
002304                  BEGIN
002305                  aFile.Become(TFile.CREATE(NIL, mainHeap, CONCAT(volumePrefix, '$T'), ''));
002306                  IF aFile.Exists(error) THEN
002307                      aFile.Rename(error, volumePrefix);
002308                  END;
002309
002310              aFile.Free;
002311
002312              SELF.files.saveExists := error <= 0;
002313              END
002314          ELSE
002315              SELF.files.saveExists := FALSE;
002316
002317          IF process.AbortRequest THEN
002318              BEGIN
002319              error := erAborted;
002320              GOTO 1;
002321              END;
002322
002323            {Try to open suspend files first, THEN the save file, THEN blank stationery}
002324          IF SELF.files.shouldSuspend THEN
002325              SELF.OpenSuspended(error, wmgrID)
002326          ELSE    {don't even try the suspend file}
002327              error := erNameNotFound;
002328
002329          IF error > 0 THEN
002330              IF error <> erAborted THEN
002331                  IF SELF.files.saveExists THEN {won't even try this if we don't create save files}
002332                      SELF.OpenSaved(error, wmgrID)
002333                  ELSE
002334                      SELF.OpenBlank(error, wmgrID)
002335              ELSE
002336                  openedSuspended := TRUE
002337          ELSE
002338              openedSuspended := TRUE;
002339
002340          IF error <= 0 THEN
002341              BEGIN
002342              SELF.dataSegment.changes := 0;
002343
002344              window := SELF.dataSegment.preludePtr^.docDirectory.window;
002345              SELF.window := window;
002346
002347              window.SetWmgrId(wmgrID); {changes the wmgrId of the window and the port of the panes}
```

```
002348
002349              pWindow := POINTER(wmgrID);
002350              SetFldrRefCon(pWindow, ORD(SELF));
002351
002352              docList.InsLast(SELF);
002353              END
002354          ELSE
002355          IF NOT openedSuspended THEN
002356              SELF.KillSegments(1, maxSegments);  {*** Good idea?}
002357  1:
002358          {$IFC fTrace}EP;{$ENDC}
002359      END;
002360      {$S SgABCres}
002361
002362
002363      {$S sCldInit}
002364      PROCEDURE {TDocManager.}OpenBlank{(VAR error: INTEGER; wmgrID: TWindowID)};
002365          LABEL 1;
002366          VAR heapSize:       LONGINT;
002367              heapStart:      LONGINT;
002368              docHeap:        THeap;
002369              prPrfAlias:     TPrPrfAlias;
002370              objCount:       INTEGER;
002371              docWindow:      TWindow;
002372              docDirectory:   TDocDirectory;
002373
002374        PROCEDURE CheckAbort;
002375        BEGIN
002376            IF process.AbortRequest THEN
002377                BEGIN
002378                error := erAborted;
002379                GOTO 1;
002380                END;
002381        END;
002382
002383      BEGIN
002384          {$IFC fTrace}BP(7);{$ENDC}
002385          heapSize := SELF.DfltHeapSize;
002386          SELF.MakeSegments(error, 0, heapSize + SIZEOF(TPrelude));
002387
002388          IF error <= 0 THEN
002389              BEGIN
002390              heapStart := ORD(SELF.dataSegment.preludePtr) + SIZEOF(TPrelude);
002391
002392              docHeap := POINTER(ORD(HzInit(POINTER(heapStart), POINTER(heapStart+heapSize),
002393                                            NIL, LIntDivInt(heapSize, 10), 0, @ExpandHeap,
002394                                            POINTER(procNil), POINTER(procNil), POINTER(procNil))));
002395                                  {*** DANGER ***}
```

```
002396                {@ExpandHeap is a pointer outside the data segment}
002397                {TDocManager.Assimilate must guarantee its accuracy}
002398
002399            CheckAbort;
002400            PrPrfDefault(prPrfAlias.prPrf);
002401            WITH SELF.dataSegment.preludePtr^ DO
002402                BEGIN
002403                password := 25376; {*** temporary ***}
002404                version := 1; {*** should be this software's version ***}
002405                country := countryCode;
002406                language := countryCode; {*** same as country code? ***}
002407                preludeSize := SIZEOF(TPrelude);
002408                printPref := prPrfAlias.reserve;
002409                END;
002410            SELF.docHeap := docHeap;
002411            docWindow := SELF.NewWindow(docHeap, wmgrID);
002412            docDirectory := TDocDirectory.CREATE(NIL, docHeap, docWindow, myWorld);
002413
002414            SELF.dataSegment.preludePtr^.docDirectory := docDirectory;
002415            docWindow.BlankStationery;
002416            CheckAbort;
002417            {$IFC fDbgABC}
002418        (*  docWindow.CheckPanels;*** Should check that union of panel rects = window rect ***)
002419            {$ENDC}
002420            END;
002421      1:  {$IFC fTrace}EP;{$ENDC}
002422      END;
002423      {$S SgABCres}
002424
002425
002426      {$S SgABCcld}
002427      PROCEDURE {TDocManager.}OpenSaved{(VAR error: INTEGER; wmgrID: TWindowID)};
002428          VAR volumePrefix:   TFilePath;
002429              aFile:          TFile;
002430              fs:             TFileScanner;
002431              fileSize:       LONGINT;
002432              preludePtr:     TPPrelude;
002433      BEGIN
002434          {$IFC fTrace}BP(7);{$ENDC}
002435          volumePrefix := SELF.files.volumePrefix;
002436          {$IFC LibraryVersion <= 20}
002437          aFile := TFile.CREATE(NIL, mainHeap, volumePrefix, '');
002438          {$ELSEC}
002439          aFile := TFile.CREATE(NIL, mainHeap, volumePrefix, SELF.files.password);
002440          {$ENDC}
002441          fs := aFile.ScannerFrom(0, [fRead]);
002442          error := fs.error;
002443
```

```
002444          IF error <= 0 THEN
002445              BEGIN
002446              fileSize := aFile.size;
002447              SELF.MakeSegments(error, 0, fileSize);
002448              IF error <= 0 THEN
002449                  BEGIN
002450                  preludePtr := SELF.dataSegment.preludePtr;
002451                  process.AbortXferSequential(xRead, POINTER(ORD(preludePtr)), fileSize, abortChunkSize, fs);
002452                  error := fs.error;
002453                  IF error <= 0 THEN
002454                      SELF.ResumeAfterOpen(error, wmgrID);
002455                  preludePtr^.docDirectory.window.changes := 0;
002456                  END;
002457              fs.Free;  {Close the file & free the TFile object}
002458              END;
002459          {$IFC fTrace}EP;{$ENDC}
002460      END;
002461      {$S SgABCres}
002462
002463
002464      {$S SgABCini}
002465      PROCEDURE {TDocManager.}OpenSuspended{(VAR error: INTEGER; wmgrID: TWindowID)};
002466          VAR volumePrefix:   TFilePath;
002467              i:              INTEGER;
002468              ldsn:           INTEGER;
002469              dsPathname:     PathName;
002470              dsRefnum:       INTEGER;
002471              memOrd:         LONGINT;
002472              preludePtr:     TPPrelude;
002473              cease:          BOOLEAN;
002474              {$IFC LibraryVersion > 20}
002475              dsPassword:     E_Name;
002476              blankPasswd:    E_Name;
002477              {$ENDC}
002478              otherError:     INTEGER;
002479      BEGIN
002480          {$IFC fTrace}BP(7);{$ENDC}
002481          IF boundDocument <> NIL THEN
002482              boundDocument.Unbind;
002483
002484          volumePrefix := SELF.files.volumePrefix;
002485          {$IFC LibraryVersion > 20}
002486          dsPassword := SELF.files.password;
002487          blankPasswd := '';
002488          {$ENDC}
002489
002490          {loop invariant: i = # datasegs already bound + 1}
002491          i := 1;
```

```
002492          REPEAT
002493              ldsn := i + docLdsn-1;
002494              dsPathname := CONCAT(volumePrefix, suspendSuffix[i]);
002495              IF currentDocument <> NIL THEN                          {*** Get around OS anomaly ***}
002496                  error := 313                                         {*** What it should return for Revert ***}
002497              ELSE                                                     {*** Remove these lines when fixed ***}
002498                  BEGIN
002499                  {$IFC LibraryVersion > 20}
002500                  Change_Password(error, dsPathname, dsPassword, blankPasswd);
002501                  {$ENDC}
002502                  Open_DataSeg(error, dsPathname, dsRefnum, memOrd, ldsn);
002503                  END;
002504
002505              IF error <= 0 THEN
002506                  BEGIN
002507                  SELF.dataSegment.refnum[i] := dsRefnum;
002508                  IF ldsn = docLdsn THEN
002509                      preludePtr := POINTER(memOrd);
002510                  SetAccess_DataSeg(error, dsRefnum, FALSE);  {Make writeable}
002511                  IF error > 0 THEN
002512                      ABCBreak('In TDocManager.OpenSuspended: SetAccess_DataSeg', error);
002513                  {$IFC LibraryVersion > 20}
002514                  Change_Password(error, dsPathname, blankPasswd, dsPassword);
002515                  IF error > 0 THEN
002516                      ABCBreak('In TDocManager.OpenSuspended: Change_Password', error);
002517                  {$ENDC}
002518                  i := i + 1;
002519                  END;
002520
002521              IF process.AbortRequest THEN
002522                  error := erAborted;
002523
002524              IF error > 0 THEN
002525                  cease := TRUE
002526              ELSE
002527                  cease := i > preludePtr^.numSegments;
002528          UNTIL cease;
002529
002530          IF error <= 0 THEN
002531              BEGIN
002532              SELF.dataSegment.preludePtr := preludePtr;
002533              boundDocument := SELF;
002534              SELF.ResumeAfterOpen(error, wmgrID);
002535              END
002536          ELSE
002537              WHILE i > 1 DO {back out by unbinding the datasegs}
002538                  BEGIN
002539                  i := i - 1;
```

```
002540                  Unbind_Dataseg(otherError, SELF.dataSegment.refnum[i]);
002541                  {$IFC fDbgABC}
002542                  IF otherError > 0 THEN
002543                      WriteLn(CHR(7), 'Error unbinding dataseg=', otherError:1);
002544                  {$ENDC}
002545                  SELF.dataSegment.refnum[i] := -1;
002546                  END;
002547          {$IFC fTrace}EP;{$ENDC}
002548      END;
002549      {$S SgABCres}
002550
002551
002552      {$S SgABCcld}
002553      PROCEDURE {TDocManager.}ResumeAfterOpen{(VAR error: INTEGER; wmgrID: TWindowID)};
002554          VAR preludePtr:     TPPrelude;
002555              docHeap:        THeap;
002556              objCount:       INTEGER;
002557      BEGIN
002558          {$IFC fTrace}BP(1);{$ENDC}
002559          error := 0;
002560
002561          preludePtr := SELF.dataSegment.preludePtr;
002562          docHeap := POINTER(ORD(preludePtr) + preludePtr^.preludeSize);
002563          SELF.docHeap := docHeap;
002564          SELF.Assimilate(error);
002565  (*****
002566          IF NOT fCheckHzOK(POINTER(ORD(docHeap)), objCount) THEN
002567              BEGIN
002568              ABCBreak('fCheckHzOK failed on suspend file: objCount', objCount);
002569              error := erInternal;
002570              END
002571          ELSE
002572              BEGIN
002573              SELF.docHeap := docHeap;
002574              SELF.Assimilate(error);
002575              END;
002576  *****)
002577          {$IFC fTrace}EP;{$ENDC}
002578      END;
002579      {$S SgABCres}
002580
002581
002582      {$S SgABCcld}
002583      PROCEDURE {TDocManager.}RevertVersion{(VAR error: INTEGER; wmgrID: TWindowID)};
002584                              { for now, must be the active window to do this }
002585          VAR dontCare:   BOOLEAN;
002586      BEGIN
002587          {$IFC fTrace}BP(1);{$ENDC}
```

```
002588          error := 0;
002589
002590          SELF.Close(FALSE);
002591              { active/current Window/Document should have been made NIL by SELF.Close }
002592          currentDocument := SELF;
002593
002594            {We could be cleverer and reuse the old data segments, later****}
002595          allowAbort := FALSE; {no abort allowed during revert}
002596          SELF.Open(error, wmgrID, dontCare);
002597          allowAbort := TRUE;
002598          IF error > 0 THEN
002599              BEGIN
002600              {$IFC fDbgABC}
002601              ABCBreak('RevertVersion error opening document', error);
002602              {$ENDC}
002603              END
002604          ELSE
002605              BEGIN
002606              PushFocus;
002607              currentWindow := SELF.window;
002608              activeWindowID := currentWindow.wmgrID;
002609              currentWindow.Focus;
002610              currentWindow.Resize(FALSE);
002611              InvalRect(currentWindow.innerRect);
002612              currentWindow.Update(TRUE);
002613              PopFocus;
002614              END;
002615          {$IFC fTrace}EP;{$ENDC}
002616      END;
002617      {$S SgABCres}
002618
002619
002620      {$S SgABCcld}
002621      PROCEDURE {TDocManager.}SaveVersion{(VAR error: INTEGER; volumePrefix: TFilePath; andContinue: BOOLEAN)};
002622          VAR tmpFile:        TFile;
002623              fs:             TFileScanner;
002624              saveFile:       TFile;
002625              localError:     INTEGER;
002626      BEGIN
002627          {$IFC fTrace}BP(7);{$ENDC}
002628          error := 0;
002629          SELF.dataSegment.preludePtr^.docDirectory.window := SELF.window;   {Just in case it somehow changed}
002630          IF NOT andContinue THEN
002631              {*** Revert to one pane per panel scrolled to the beginning & no (or standard) selection***};
002632
002633          IF process.AbortRequest THEN
002634              error := erAborted
002635          ELSE
```

```
002636              BEGIN
002637             {SELF.ReleaseDiskSpace...;   *** TO DO **ONLY IF** WE CAN'T GET ENOUGH SPACE WITHOUT ***}
002638              IF process.AbortRequest THEN
002639                  error := erAborted
002640              ELSE
002641                  BEGIN
002642                  {$IFC LibraryVersion <= 20}
002643                  tmpFile := TFile.CREATE(NIL, mainHeap, CONCAT(volumePrefix, '$T'), '');
002644                  {$ELSEC}
002645                  tmpFile := TFile.CREATE(NIL, mainHeap, CONCAT(volumePrefix, '$T'), SELF.files.password);
002646                  {$ENDC}
002647                  fs := tmpFile.ScannerFrom(0, [fWrite]);
002648                  error := fs.error;
002649
002650                  IF error <= 0 THEN
002651                      IF process.AbortRequest THEN
002652                          error := erAborted;
002653
002654                  IF error > 0 THEN
002655                      BEGIN
002656                      tmpFile.Delete(localError);
002657                      fs.Free;
002658                      END
002659                  ELSE
002660                      BEGIN
002661                      process.AbortXferSequential(xWrite, POINTER(ORD(SELF.dataSegment.preludePtr)),
002662                                                  SELF.dataSegment.preludePtr^.docSize, abortChunkSize, fs);
002663                      fs.Compact;
002664                      {*** we should set the logical file size to the logical EOF ***}
002665
002666                      error := fs.error;
002667                      {*** Be sure buffers are flushed ***}
002668
002669                      IF error <= 0 THEN
002670                          IF process.AbortRequest THEN
002671                              error := erAborted;
002672
002673                      IF error > 0 THEN
002674                          BEGIN
002675                          {$IFC fDbgABC}
002676                          ABCbreak('In TDocManager.SaveVersion, error saving file=', error);
002677                          {$ENDC}
002678
002679                          {this is after we wrote out the file, need a wait alert if user aborted}
002680                          IF error = erAborted THEN
002681                              process.BeginWait(phAborting);
002682                          tmpFile.Delete(localError);
002683
```

```
002684                          {$IFC fDbgABC}
002685                          IF localError > 0 THEN
002686                              ABCbreak('In TDocManager.SaveVersion, error deleting file=', localError);
002687                          {$ENDC}
002688
002689                          process.EndWait;
002690                          fs.Free;
002691                          END
002692                      ELSE
002693                          BEGIN
002694                          fs.FreeObject; {don't free tmpFile yet}
002695                          IF SELF.files.saveExists THEN
002696                              BEGIN
002697                              {$IFC LibraryVersion <= 20}
002698                              saveFile := TFile.CREATE(NIL, mainHeap, volumePrefix, '');
002699                              {$ELSEC}
002700                              saveFile := TFile.CREATE(NIL, mainHeap, volumePrefix, SELF.files.password);
002701                              {$ENDC}
002702                              saveFile.Delete(localError);
002703                              saveFile.Free;
002704                              END;
002705                          SELF.files.saveExists := TRUE;
002706                          tmpFile.Rename(localError, volumePrefix);
002707
002708                          {$IFC fDbgABC}
002709                          IF localError > 0 THEN
002710                              ABCbreak('In TDocManager.SaveVersion, error renaming file=', localError);
002711                          {$ENDC}
002712
002713                          tmpFile.Free;
002714                          SELF.window.changes := 0;
002715                          END;
002716                      END;
002717                  END;
002718              END;
002719          {$IFC fTrace}EP;{$ENDC}
002720      END;
002721      {$S SgABCres}
002722
002723
002724      {$S sStartup}
002725      PROCEDURE {TDocManager.}SetSegSize{(VAR error: INTEGER; minSize, maxExcess: LONGINT)};
002726          {Make the memory and disk size of the virtual data segment be at least as indicated, and leave
002727           some excess, but no more than the maximum indicated.  Update docSize, numSegments, and the
002728           refnum table.  Assumptions:
002729              The virtual data segment exists and is open and bound.
002730              It has at least one real data segment, and has a valid heap that fits in the
002731                  lesser of the current diskSize and the new diskSize.
```

```
002732          All LONG parameters are rounded up IF necessary to a multiple of 512 before they are used.}
002733
002734          VAR preludePtr:         TPPrelude;
002735              dsInfo:             DsInfoRec;
002736              oldMemSize:         LONGINT;
002737              newSize:            LONGINT;
002738              newSegments:        INTEGER;
002739              newSegSize:         LONGINT;
002740              temp:               LONGINT;
002741              sched_err:          INTEGER;
002742      BEGIN
002743          {$IFC fTrace}BP(7);{$ENDC}
002744          error := 0;
002745
002746          minSize := LIntMulInt(LIntDivInt(minSize + 511, 512), 512);
002747          maxExcess := LIntMulInt(LIntDivInt(maxExcess + 511, 512), 512);
002748
002749          preludePtr := SELF.dataSegment.preludePtr;
002750
002751          WITH preludePtr^ DO
002752              BEGIN
002753              Info_DataSeg(error, SELF.dataSegment.refNum[numSegments], dsInfo);
002754              IF error > 0 THEN
002755                  ABCBreak('SetSegSize: Info_Dataseg', error);
002756
002757              oldMemSize := dsInfo.mem_size + (maxSegSize*(numSegments-1));
002758
002759              IF (oldMemSize < minSize) OR (oldMemSize > minSize + maxExcess) THEN
002760                  {need to adjust the segment size}
002761                  BEGIN
002762                  newSize := minSize + maxExcess;
002763                  newSegments := LIntDivLInt(newSize + maxSegSize - 1, maxSegSize);
002764
002765                  {$IFC fDbgABC}
002766                  IF (numSegments < 1) OR (numSegments > maxSegments) THEN
002767                      ABCBreak('SetSegSize: numSegments NOT IN 1..maxSegments', numSegments);
002768                  IF (newSegments < 1) OR (newSegments > maxSegments) THEN
002769                      ABCBreak('SetSegSize: newSegments NOT IN 1..maxSegments', newSegments);
002770                  {$ENDC}
002771
002772                  IF numSegments > newSegments THEN
002773                      {kill off whole segments we don't need anymore}
002774                      SELF.KillSegments(newSegments + 1, numSegments)
002775                  ELSE
002776                  IF numSegments < newSegments THEN
002777                      SELF.MakeSegments(error, numSegments, newSize);
002778                          {this sets all the segment sizes correctly}
002779
```

```
002780                  {resize the new last segment}
002781                  newSegSize := newSize - (maxSegSize*(newSegments-1));
002782                                  {total doc size - size of all segments before last one}
002783
002784                  Info_DataSeg(error, SELF.dataSegment.refNum[newSegments], dsInfo);
002785                  IF error > 0 THEN
002786                      ABCBreak('SetSegSize: Info_Dataseg', error);
002787
002788                  WITH dsInfo DO
002789                      BEGIN
002790                      Sched_Class(sched_err, FALSE);
002791                      Size_Dataseg(error, SELF.dataSegment.refnum[newSegments],
002792                                  newSegSize-mem_size, temp, newSegSize-disc_size, temp);
002793                      Sched_Class(sched_err, TRUE);
002794                      END;
002795
002796                  {$IFC fDbgABC}
002797                  IF fExperimenting THEN
002798                      BEGIN
002799                      WriteLn('In SetSegSize: newSize=', newSize:1, '  newSegments=', newSegments:1);
002800                      WITH dsInfo DO
002801                          WriteLn('newSegSize=',newSegSize:1, '  mem_size=', mem_size:1,
002802                                  '  disc_size=', disc_size:1);
002803                      END;
002804                  {$ENDC}
002805
002806                  IF error > 0 THEN
002807                      BEGIN
002808                      {$IFC fDbgABC}
002809                      WriteLn('In SetSegSize: newSize=', newSize:1, '  newSegments=', newSegments:1);
002810                      WITH dsInfo DO
002811                          WriteLn('newSegSize=',newSegSize:1, '  mem_size=', mem_size:1,
002812                                  '  disc_size=', disc_size:1);
002813                      {$ENDC}
002814                      ABCBreak('In TDocManager.SetSegSize: Size_Dataseg', error);
002815                      END
002816                  ELSE
002817                      BEGIN
002818                      docSize := newSize;
002819                      numSegments := newSegments;
002820                      END;
002821                  END;
002822              END;
002823          {$IFC fTrace}EP;{$ENDC}
002824      END;
002825      {$S SgABCres}
002826
002827
```

```
002828      {$S SgABCcld}
002829      PROCEDURE {TDocManager.}Suspend{(VAR error: INTEGER)};
002830          LABEL 1;
002831          VAR lastSegClosed:  INTEGER;
002832              osErr:          INTEGER;
002833  (********** THESE VARIABLES ARE NEEDED ONLY IF SUSPEND IS ABORTABLE
002834              volumePrefix:   TFilePath;
002835              ldsn:           INTEGER;
002836              dsPathname:     PathName;
002837              dsRefnum:       INTEGER;
002838              memOrd:         LONGINT;
002839              reopenedSeg:    INTEGER;
002840  **********)
002841      BEGIN
002842          {$IFC fTrace}BP(7);{$ENDC}
002843          {$IFC fDbgABC}
002844          IF SELF <> boundDocument THEN
002845              ABCBreak('Suspend not-bound document', error);
002846          {$ENDC}
002847
002848          SELF.dataSegment.preludePtr^.docDirectory.window := SELF.window; {In case it somehow changed}
002849
002850          error := 0;     {*** error return here not very meaningful yet ***}
002851
002852          FOR lastSegClosed := 1 TO SELF.dataSegment.preludePtr^.numSegments DO
002853              BEGIN
002854              Close_Dataseg(osErr, SELF.dataSegment.refnum[lastSegClosed]);
002855              LatestError(osErr, error);
002856              SELF.dataSegment.refnum[lastSegClosed] := -1;
002857
002858  (********** DOES IT MAKE ANY SENSE FOR SUSPEND TO BE ABORTABLE ???? **********
002859              IF process.AbortRequest THEN
002860                  BEGIN
002861                  volumePrefix := SELF.files.volumePrefix;
002862                  FOR reopenedSeg := 1 TO lastSegClosed DO
002863                      BEGIN
002864                      ldsn := reopenedSeg + docLdsn-1;
002865                      dsPathname := CONCAT(volumePrefix, suspendSuffix[reopenedSeg]);
002866                      Open_DataSeg(osErr, dsPathname, dsRefnum, memOrd, ldsn);
002867                      LatestError(osErr, error);
002868                      IF osErr <= 0 THEN
002869                          BEGIN
002870                          SELF.dataSegment.refnum[reopenedSeg] := dsRefnum;
002871                          SetAccess_DataSeg(osErr, dsRefnum, FALSE);  {Make writeable}
002872                          IF osErr > 0 THEN
002873                              ABCBreak('ReopenDatasegs, SetAccess_DataSeg', osErr);
002874                          END
002875                      ELSE
```

```
002876                          GOTO 1;
002877                      END;
002878                  IF error <= 0 THEN
002879                      error := erAborted;
002880                  GOTO 1;
002881                  END;
002882  **********)
002883              END;
002884          SELF.dataSegment.changes := 0;
002885          boundDocument := NIL;
002886      1:  {$IFC fTrace}EP;{$ENDC}
002887      END;
002888      {$S SgABCres}
002889
002890
002891      {$S SgABCcld}
002892      PROCEDURE {TDocManager.}Unbind;
002893          VAR error:  INTEGER;
002894              i:      INTEGER;
002895      BEGIN
002896          {$IFC fTrace}BP(7);{$ENDC}
002897          IF SELF = boundDocument THEN
002898              BEGIN
002899  (***** See how things work without this check
002900              {$IFC fDbgABC}
002901              IF SELF = currentDocument THEN
002902                  ABCBreak('Unbind currentDocument', ORD(SELF));
002903              {$ENDC}
002904  *****)
002905
002906              FOR i := 1 TO SELF.dataSegment.preludePtr^.numSegments DO
002907                  BEGIN
002908                  Unbind_DataSeg(error, SELF.dataSegment.refnum[i]);
002909                  IF error > 0 THEN
002910                      ABCBreak('Unbind_DataSeg', error);
002911                  END;
002912
002913              boundDocument := NIL;
002914              END;
002915          {$IFC fTrace}EP;{$ENDC}
002916      END;
002917
002918
002919      {$S sRes}
002920      FUNCTION  {TDocManager.}WindowWithId{(wmgrID: TWindowID): TWindow};
002921      BEGIN
002922          {$IFC fTrace}BP(7);{$ENDC}
002923          IF SELF.window.wmgrID = wmgrID THEN
```

```
002924              WindowWithId := SELF.window
002925          ELSE
002926              WindowWithId := NIL;
002927          {$IFC fTrace}EP;{$ENDC}
002928      END;
002929
002930
002931  {$S SgABCini}
002932  END;
002933
002934
002935  METHODS OF TClipboard;
002936
002937
002938      {$S SgABCini}
002939      FUNCTION  {TClipboard.}CREATE{(object: TObject; heap: THeap): TClipboard};
002940      BEGIN
002941          {$IFC fTrace}BP(7);{$ENDC}
002942          IF object = NIL THEN
002943              object := NewObject(heap, THISCLASS);
002944          SELF := TClipboard(TDocManager.CREATE(object, heap, '--CLIPBOARD'));
002945          WITH SELF DO
002946              BEGIN
002947              hasView := FALSE;
002948              hasPicture := FALSE;
002949              hasUniversalText := FALSE;
002950              hasIcon := FALSE;
002951              cuttingTool := 0;
002952              cuttingProcessID := 0;
002953              clipCopy := NIL;
002954              END;
002955          {$IFC fTrace}EP;{$ENDC}
002956      END;
002957      {$S SgABCres}
002958
002959
002960      {$IFC fDebugMethods}
002961      {$S SgABCdbg}
002962      PROCEDURE {TClipboard.}Fields{(PROCEDURE Field(nameAndType: S255))};
002963      BEGIN
002964          TDocManager.Fields(Field);
002965          Field('hasView: BOOLEAN');
002966          Field('hasPicture: BOOLEAN');
002967          Field('hasUniversalText: BOOLEAN');
002968          Field('hasIcon: BOOLEAN');
002969          Field('cuttingTool: LONGINT');
002970          Field('cuttingProcessID: LONGINT');
002971          Field('clipCopy: TFileScanner;');
```

```
002972      END;
002973      {$S SgABCres}
002974      {$ENDC}
002975
002976      {$S sCut}
002977      PROCEDURE {TClipboard.}AboutToCut;
002978      BEGIN
002979          {$IFC fTrace}BP(7);{$ENDC}
002980          InheritScrap(TRUE);
002981          {$IFC fTrace}EP;{$ENDC}
002982      END;
002983
002984
002985      {$S sCut}
002986      PROCEDURE {TClipboard.}BeginCut;
002987          LABEL   1;
002988
002989          VAR heap:       THeap;
002990              window:     TWindow;
002991              panel:      TPanel;
002992              view:       TView;
002993              selection:  TSelection;
002994              error:      INTEGER;
002995      BEGIN
002996          {$IFC fTrace}BP(7);{$ENDC}
002997          IF boundClipboard = NIL THEN
002998              boundClipboard := SELF
002999          ELSE
003000              ABCBreak('BeginCut: Clipboard already bound', 0);
003001
003002          EraseScrapData(error);
003003          IF error > 0 THEN
003004              BEGIN
003005              ABCBreak('EraseScrapData', error);
003006              BackOutOfScrap;
003007              {need to put up alert that cut was not put into scrap and pass this info back up the ladder}
003008              GOTO 1;
003009              END;
003010
003011          {Obtain write access}
003012          StartPutScrap(error);
003013          IF error > 0 THEN
003014              BEGIN
003015              ABCBreak('StartPutScrap', error);
003016              BackOutOfScrap;
003017              {need to put up alert that cut was not put into scrap and pass this info back up the ladder}
003018              GOTO 1;
003019              END;
```

```
003020
003021          {Find out where the Clipboard heap is}
003022          heap := POINTER(ORD(hzOfScrap));
003023          SELF.docHeap := heap;
003024
003025          {Create a standard window onto the Clipboard}
003026          window := SELF.NewWindow(heap, ORD(scrapFolder));
003027          SELF.window := window;
003028          panel := TPanel.CREATE(NIL, heap, window, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
003029
003030          {Create a dummy view to be replaced by the application's view}
003031          view := panel.NewStatusView(NIL, zeroLRect);
003032
003033          clipPrintPref := boundDocument.dataSegment.preludePtr^.printPref;
003034  1:
003035          {$IFC fTrace}EP;{$ENDC}
003036      END;
003037
003038
003039      {$S sPaste}
003040      PROCEDURE {TClipboard.}Bind;
003041          VAR which:          ScrapType;
003042              what:           TH;
003043              docDirectory:   TDocDirectory;
003044              olderVersion:   BOOLEAN;
003045              newerVersion:   BOOLEAN;
003046              error:          INTEGER;
003047
003048          PROCEDURE CopyScrap;
003049              VAR aFile:      TFile;
003050                  fs:         TFileScanner;
003051                  dsInfo:     DsInfoRec;
003052          BEGIN
003053              aFile := TFile.CREATE(NIL, mainHeap, 'TKScrapCopy', '');
003054              fs := aFile.Scanner;
003055              SELF.clipCopy := fs;
003056              Info_Dataseg(error, DSegOfScrap, dsInfo);
003057              {$IFC fDbgABC}
003058              IF error > 0 THEN
003059                  ABCbreak('CopyScrap: error from Info_Dataseg', error);
003060              {$ENDC}
003061
003062              WITH dsInfo DO
003063                  fs.XferSequential(xWrite, Ptr(AddrOfScrapDSeg), mem_size);
003064          END;
003065
003066      BEGIN
003067          {$IFC fTrace}BP(7);{$ENDC}
```

```
003068          IF boundClipboard <> SELF THEN
003069              BEGIN
003070              IF boundClipboard <> NIL THEN
003071                  boundClipboard.Unbind;
003072              boundClipboard := SELF;
003073
003074              {Open the clipboard data segment}
003075              StartGetScrap(error);
003076              IF error > 0 THEN
003077                  BEGIN
003078                  ABCBreak('StartGetScrap', error);
003079                  BackOutOfScrap;
003080                  {need to put up alert that scrap cannot be bound and pass this info back up the ladder}
003081                  END
003082
003083              ELSE
003084                  BEGIN
003085                  {Obtain write access}
003086                  SetAccess_DataSeg(error, DSegOfScrap, FALSE);
003087                  IF error > 0 THEN
003088                      ABCBreak('SetAccess_DataSeg', error);
003089
003090                  {Find out what is there to be pasted}
003091                  GetScrap(which, what);
003092
003093                  SELF.window := NIL;
003094
003095  {$IFC LibraryVersion > 20}
003096                  IF scrapProcess = myProcessID THEN
003097                      IF which = scrapRef THEN
003098                          BEGIN
003099                          which := toolKitType;
003100                          what := Pointer(Ord(GetFldrRefCon(scrapFolder)));
003101                          END;
003102  {$ENDC}
003103                  IF which = toolKitType THEN
003104                      BEGIN
003105                      docDirectory := POINTER(ORD(what));
003106  (**)
003107  (*^*)               IF scrapProcess <> myProcessID THEN  {Don't waste time checking if I put it there myself}
003108                          IF NeedConversion(docDirectory.classWorld, olderVersion, newerVersion) THEN
003109                              BEGIN
003110                              CopyScrap;
003111                              {*** Should defer until app likes selection class ***}
003112                              (*** ClaimScrap; ***) (*^*)
003113
003114                              ConvertHeap(POINTER(ORD(HzOfScrap)), docDirectory.classWorld);
003115                              docDirectory.Adopt; (*^*)
```

```
003116                              END;
003117  (**)
003118                          SELF.window := docDirectory.window;
003119                          END;
003120
003121                  {Record attributes of the clipboard data that the application might want to inquire about}
003122                  SELF.Inspect;
003123                  END;
003124              END;
003125          {$IFC fTrace}EP;{$ENDC}
003126      END;
003127
003128
003129      {$S sCut}
003130      PROCEDURE {TClipboard.}CommitCut;
003131      BEGIN
003132          {$IFC fTrace}BP(7);{$ENDC}
003133          AcceptInheritScrap;
003134          {$IFC fTrace}EP;{$ENDC}
003135      END;
003136
003137
003138      {$S sCut}
003139      PROCEDURE {TClipboard.}EndCut;
003140          VAR window:         TWindow;
003141              clipSel:        TSelection;
003142              docDirectory:   TDocDirectory;
003143              error:          INTEGER;
003144      BEGIN
003145          {$IFC fTrace}BP(7);{$ENDC}
003146          window := SELF.window;
003147          clipSel := window.selectPanel.selection;
003148
003149          IF clipSel.kind = nothingKind THEN
003150              BEGIN
003151              {$IFC fDbgABC}
003152              ABCBreak('No selection in Clipboard at EndCut', 0);
003153              {$ENDC}
003154              BackOutOfScrap;
003155              {need to put up an alert and pass info up the ladder}
003156              END
003157          ELSE
003158              BEGIN
003159              {Display the Clipboard}
003160              PushFocus;
003161              window.Focus;
003162              window.Refresh([rErase, rFrame, rBackground, rDraw], hNone);
003163              PopFocus;
```

```
003164
003165              {Inform others of what TWindow is there to be pasted}
003166              docDirectory := TDocDirectory.CREATE(NIL, SELF.docHeap, window, myWorld);
003167              PutScrap(toolKitType, POINTER(ORD(docDirectory)), error);
003168
003169              {Record attributes of the clipboard data that the application might want to inquire about}
003170              SELF.Inspect;
003171
003172              {Relinquish write access}
003173              EndPutScrap(error);
003174              IF error > 0 THEN
003175                  ABCBreak('EndPutScrap', error);
003176              END;
003177
003178          SELF.window := NIL;
003179          boundClipboard := NIL;
003180          {$IFC fTrace}EP;{$ENDC}
003181      END;
003182
003183
003184      {$S sCldInit}
003185      PROCEDURE {TClipboard.}Inspect;
003186          VAR which:      ScrapType;
003187              what:       TH;
003188              pic:        PicHandle;
003189      BEGIN
003190          {$IFC fTrace}BP(7);{$ENDC}
003191    {$H-} SELF.docHeap := POINTER(ORD(HzOfScrap)); {$H+}
003192          GetScrap(which, what);
003193          SELF.hasView := which = toolKitType;
003194    {$H-} GetGrScrap(pic); {$H+}
003195          SELF.hasPicture := pic <> NIL;
003196          SELF.hasUniversalText := (scrapCs IN currScrapSet);
003197  {$IFC LibraryVersion > 20}
003198          SELF.hasIcon := which = scrapRef;
003199  {$ENDC}
003200          SELF.cuttingProcessID := scrapProcess;
003201    {$H-} SELF.cuttingTool := ToolOfProcess(scrapProcess); {$H+}
003202          {$IFC fTrace}EP;{$ENDC}
003203      END;
003204
003205
003206      {$S SgABCcld}
003207      PROCEDURE {TClipboard.}Publicize;
003208          VAR window:             TWindow;
003209              panel:              TPanel;
003210              pane:               TPane;
003211              viewExtentLRect:    LRect;
```

```
003212              info:               WindowInfo;
003213              picLRect:           LRect;
003214              tempHeap:           THeap;
003215              picRect:            Rect;
003216              tempPad:            TPad;
003217              pic:                PicHandle;
003218              error:              INTEGER;
003219      BEGIN
003220          {$IFC fTrace}BP(7);{$ENDC}
003221          IF scrapProcess = myProcessID THEN
003222              BEGIN
003223              SELF.Bind;
003224
003225              window := SELF.window;
003226              IF window <> NIL THEN {LSR}
003227                  BEGIN {LSR}
003228                  panel := TPanel(window.panels.First);
003229                  pane := TPane(panel.panes.First);
003230                  viewExtentLRect := window.selectPanel.view.extentLRect;
003231
003232                  {Let the Window Manager have a picture to display while inactive [if open]}
003233                  GetWindInfo(POINTER(window.wmgrID), info);
003234                  IF info.visible THEN
003235                      window.StashPicture(hNone);
003236
003237                  {Let others have a picture to paste}
003238                  noPad.RectToLRect(hugeRect, picLRect);
003239                  IF SectLRect(viewExtentLRect, picLRect, picLRect) AND NOT EmptyLRect(picLRect) THEN
003240                      BEGIN
003241                      GetHeap(tempHeap);
003242                      SetHeap(POINTER(ORD(HzOfScrap)));
003243
003244                      {Before calling Focus, set up everything for unclipped drawing of the view}
003245                      tempPad := TPad.CREATE(NIL, mainHeap, hugeRect, picLRect, screenRes,
003246                                             screenRes, thePort);
003247                      tempPad.LRectToRect(picLRect, picRect);
003248                      RectRgn(altVisRgn, picRect);
003249                      useAltVisRgn := TRUE;           { enable clipping to whole picture }
003250
003251                      {Focus on the Clipboard}
003252                      PushFocus;
003253                      tempPad.Focus;
003254                      focusArea := NIL;               {To trap illegal attempts to Push/PopFocus during
003255                                                       TView.Draw}
003256
003257                      {Generate the Universal Picture}
003258                      pic := OpenPicture(picRect);
003259
```

```
003260                  genClipPic := TRUE;             { enable putting comments into picture }
003261                  PicComment(cPicGeDwg, 0, NIL);  { needed for pasting into LisaDraw }
003262                  PicGrpBegin;                    { every LisaDraw picture from other apps is a group }
003263                  panel.view.Draw;                { tell the application to draw now }
003264                  PicGrpEnd;
003265
003266                  ClosePicture;
003267
003268                  {Put it in the Clipboard}
003269                  PutGrScrap(pic, error);
003270                  IF error > 0 THEN
003271                      ABCBreak('PutGrScrap', error);
003272
003273                  {Generate the Universal Text}
003274                  panel.view.CreateUniversalText;
003275
003276                  {Unravel}
003277                  genClipPic := FALSE;            { disable putting comments into picture }
003278                  useAltVisRgn := FALSE;          { disable clipping to whole window }
003279                  PopFocus;
003280                  tempPad.Free;
003281                  SetHeap(tempHeap);
003282                  END;
003283              END; {LSR}
003284
003285          SELF.Unbind;
003286          END;
003287      {$IFC fTrace}EP;{$ENDC}
003288  END;
003289
003290
003291  {$S sPaste}
003292  PROCEDURE {TClipboard.}Unbind;
003293      VAR error:      INTEGER;
003294
003295      PROCEDURE RestoreScrap;
003296          VAR fs: TFileScanner;
003297      BEGIN
003298          fs := SELF.clipCopy;
003299          IF fs <> NIL THEN
003300              BEGIN
003301              fs.XferRandom(xRead, Ptr(AddrOfScrapDSeg), fs.actual, fAbsolute, 0);
003302              fs.Free;
003303              SELF.clipCopy := NIL;
003304              END;
003305      END;
003306
003307  BEGIN
```

```
003308          {$IFC fTrace}BP(7);{$ENDC}
003309          IF SELF = boundClipboard THEN
003310              BEGIN
003311              RestoreScrap;
003312
003313              {$IFC fDbgABC}
003314              IF SELF = currentDocument THEN
003315                  ABCBreak('TClipboard.Unbind currentDocument', ORD(SELF));
003316              {$ENDC}
003317
003318              boundClipboard := NIL;
003319
003320              {Relinquish access}
003321              SELF.window := NIL;
003322              EndGetScrap(error);
003323
003324              IF error > 0 THEN
003325                  ABCBreak('EndGetScrap', error);
003326              END;
003327          {$IFC fTrace}EP;{$ENDC}
003328      END;
003329
003330
003331      {$S sCut}
003332      FUNCTION  {TClipboard.}UndoCut{: BOOLEAN};
003333          VAR clipErr:    INTEGER;
003334      BEGIN
003335          {$IFC fTrace}BP(7);{$ENDC}
003336          UndoInheritScrap(clipErr);
003337          SELF.Inspect;       {so app can inquire}
003338
003339  (*      IF (clipErr <= 0) AND SELF.hasView THEN  * WRONG BECAUSE SELF.window MAY BELONG TO ANOTHER TK APP *
003340              BEGIN
003341              SELF.Bind;
003342              SELF.window.Resize(FALSE); {in case clipboard resized between the cut and the undo-cut}
003343              SELF.Unbind;
003344              END;
003345  *)
003346          UndoCut := clipErr <= 0;
003347          {$IFC fTrace}EP;{$ENDC}
003348      END;
003349
003350
003351  {$S SgABCini}
003352  END;
003353
003354
003355  METHODS OF TCommand;
```

```
003356
003357
003358      {$S sCommand}
003359      FUNCTION  {TCommand.}CREATE{(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
003360                                   itsImage: TImage; isUndoable: BOOLEAN; itsRevelation: TRevelation): TCommand};
003361          VAR cmdPhase:   TCmdPhase;
003362      BEGIN
003363          {$IFC fTrace}BP(6);{$ENDC}
003364          IF object = NIL THEN
003365              object := NewObject(heap, THISCLASS);
003366          SELF := TCommand(object);
003367
003368          WITH SELF DO
003369              BEGIN
003370              cmdNumber := itsCmdNumber;
003371              image := itsImage;
003372              undoable := isUndoable;
003373              doing := FALSE;
003374              revelation := itsRevelation;
003375              FOR cmdPhase := doPhase TO redoPhase DO
003376                  BEGIN
003377                  unHiliteBefore[cmdPhase] := TRUE;
003378                  hiliteAfter[cmdPhase] := TRUE;
003379                  END;
003380              END;
003381          {$IFC fTrace}EP;{$ENDC}
003382      END;
003383
003384
003385      {$IFC fDebugMethods}
003386      {$S SgABCdbg}
003387      PROCEDURE {TCommand.}Fields{(PROCEDURE Field(nameAndType: S255))};
003388      BEGIN
003389          Field('cmdNumber: INTEGER');
003390          Field('image: TImage');
003391          Field('undoable: BOOLEAN');
003392          Field('doing: BOOLEAN');
003393          Field('revelation: Byte');
003394          Field('unHiliteBefore: ARRAY[0..2] OF BOOLEAN');
003395          Field('hiliteAfter: ARRAY[0..2] OF BOOLEAN');
003396          Field('');
003397      END;
003398      {$S SgABCres}
003399      {$ENDC}
003400
003401
003402      {$S sCommand}
003403      PROCEDURE {TCommand.}Commit;
```

```
003404      BEGIN
003405          {$IFC fTrace}BP(7);{$ENDC}
003406          {$IFC fTrace}EP;{$ENDC}
003407      END;
003408
003409
003410      {$S sFilter}
003411      PROCEDURE {TCommand.}EachVirtualPart{(PROCEDURE DoToObject(filteredObj: TObject))};
003412
003413          PROCEDURE DoToFilteredObject(actualObj: TObject);
003414          BEGIN
003415              SELF.FilterAndDo(actualObj, DoToObject);
003416          END;
003417
003418      BEGIN
003419          {$IFC fTrace}BP(11);{$ENDC}
003420          IF SELF.image <> NIL THEN
003421              SELF.image.EachActualPart(DoToFilteredObject)
003422          ELSE
003423              currentWindow.EachActualPart(DoToObject);
003424          {$IFC fTrace}EP;{$ENDC}
003425      END;
003426
003427
003428      {$S sFilter}
003429      PROCEDURE {TCommand.}FilterAndDo{(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject))};
003430      BEGIN
003431          {$IFC fTrace}BP(11);{$ENDC}
003432          DoToObject(actualObj);
003433          {$IFC fTrace}EP;{$ENDC}
003434      END;
003435
003436
003437      {$S sCommand}
003438      PROCEDURE {TCommand.}Perform{(cmdPhase: TCmdPhase)};
003439      BEGIN
003440          {$IFC fTrace}BP(7);{$ENDC}
003441          {$IFC fTrace}EP;{$ENDC}
003442      END;
003443
003444
003445  {$S SgABCini}
003446  END;
003447  {$S SgABCres}
003448
003449
003450  METHODS OF TCutCopyCommand;
003451
```

```
003452
003453      {$S sCut}
003454      FUNCTION  {TCutCopyCommand.}CREATE{(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
003455                                          itsImage: TImage; isCutCmd: BOOLEAN): TCutCopyCommand};
003456      BEGIN
003457          {$IFC fTrace}BP(6);{$ENDC}
003458          IF object = NIL THEN
003459              object := NewObject(heap, THISCLASS);
003460          SELF := TCutCopyCommand(TCommand.CREATE(object, heap, itsCmdNumber, itsImage, TRUE, revealAll));
003461          SELF.isCut := isCutCmd;
003462          {$IFC fTrace}EP;{$ENDC}
003463      END;
003464
003465
003466      {$IFC fDebugMethods}
003467      {$S SgABCdbg}
003468      PROCEDURE {TCutCopyCommand.}Fields{(PROCEDURE Field(nameAndType: S255))};
003469      BEGIN
003470          SUPERSELF.Fields(Field);
003471          Field('isCut: BOOLEAN');
003472          Field('');
003473      END;
003474      {$S SgABCcld}
003475      {$ENDC}
003476
003477
003478      {$S sCut}
003479      PROCEDURE {TCutCopyCommand.}Commit;
003480      BEGIN
003481          {$IFC fTrace}BP(7);{$ENDC}
003482          clipboard.CommitCut;
003483          {$IFC fTrace}EP;{$ENDC}
003484      END;
003485
003486
003487      {$S Override}
003488      PROCEDURE {TCutCopyCommand.}DoCutCopy{(clipSelection: TSelection; deleteOriginal: BOOLEAN;
003489                                            cmdPhase: TCmdPhase)};
003490      BEGIN
003491          {$IFC fTrace}BP(7);{$ENDC}
003492          {$IFC fTrace}EP;{$ENDC}
003493      END;
003494
003495
003496      {$S sCut}
003497      PROCEDURE {TCutCopyCommand.}Perform{(cmdPhase: TCmdPhase)};
003498      BEGIN
003499          {$IFC fTrace}BP(7);{$ENDC}
```

```
003500          CASE cmdPhase OF
003501              doPhase:
003502                  BEGIN
003503                  clipboard.AboutToCut;
003504                  clipboard.BeginCut;
003505                  SELF.DoCutCopy(clipboard.window.selectPanel.selection, SELF.isCut, cmdPhase);
003506                  clipboard.EndCut;
003507                  END;
003508              undoPhase:
003509                  BEGIN
003510                  IF SELF.isCut THEN
003511                      BEGIN
003512                      IF NOT clipboard.hasView THEN
003513                          ABCbreak('undoing Cut but clipboard has no view', 0)
003514                      ELSE
003515                          BEGIN
003516                          clipboard.Bind;
003517                          IF clipboard.window = NIL THEN
003518                              ABCbreak('undoing Cut but clipboard.window = NIL', 0)
003519                          ELSE
003520                              SELF.DoCutCopy(clipboard.window.selectPanel.selection, TRUE, cmdPhase);
003521                          clipboard.Unbind;
003522                          END;
003523                      END
003524                  ELSE
003525                      SELF.DoCutCopy(NIL, FALSE, cmdPhase);
003526                  IF NOT clipboard.UndoCut THEN
003527                      BEGIN
003528                      {$IFC fDbgABC}
003529                      ABCbreak('clipboard.UndoCut returns FALSE', 0);
003530                      {$ENDC}
003531                      END;
003532                  END;
003533              redoPhase:
003534                  BEGIN
003535                  IF NOT clipBoard.UndoCut THEN
003536                      BEGIN
003537                      ABCbreak('clipboard.UndoCut returns FALSE', 0);
003538                      END
003539                  ELSE
003540                      BEGIN
003541                      clipboard.Bind;
003542                      IF NOT clipboard.hasView THEN
003543                          ABCbreak('re-doing Cut/Copy but clipboard has no view', 0)
003544                      ELSE
003545                          SELF.DoCutCopy(clipboard.window.selectPanel.selection, SELF.isCut, cmdPhase);
003546                      clipboard.Unbind;
003547                      END;
```

```
003548                END;
003549            END;
003550        {$IFC fTrace}EP;{$ENDC}
003551    END;
003552
003553
003554  {$S SgABCini}
003555  END;
003556  {$S SgABCres}
003557
003558
003559  METHODS OF TPasteCommand;
003560
003561
003562      {$S sPaste}
003563      FUNCTION  {TPasteCommand.}CREATE{(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
003564                                        itsImage: TImage): TPasteCommand};
003565      BEGIN
003566          {$IFC fTrace}BP(6);{$ENDC}
003567          IF object = NIL THEN
003568              object := NewObject(heap, THISCLASS);
003569          SELF := TPasteCommand(TCommand.CREATE(object, heap, itsCmdNumber, itsImage, TRUE, revealAll));
003570          {$IFC fTrace}EP;{$ENDC}
003571      END;
003572
003573
003574      {$S Override}
003575      PROCEDURE {TPasteCommand.}DoPaste{(clipSelection: TSelection; pic: PicHandle; cmdPhase: TCmdPhase)};
003576      BEGIN
003577          {$IFC fTrace}BP(7);{$ENDC}
003578          {$IFC fTrace}EP;{$ENDC}
003579      END;
003580
003581
003582      {$S sPaste}
003583      PROCEDURE {TPasteCommand.}Perform{(cmdPhase: TCmdPhase)};
003584      VAR window:     TWindow;
003585          pic:        PicHandle;
003586          selection:  TSelection;
003587      BEGIN
003588          {$IFC fTrace}BP(7);{$ENDC}
003589          CASE cmdPhase OF
003590              doPhase, redoPhase:
003591                  IF NOT (clipboard.hasPicture OR clipboard.hasView OR clipboard.hasUniversalText) THEN
003592                      IF currScrapSet = [] THEN
003593                          process.Stop(phNoClip)
003594                      ELSE
003595                          process.Stop(phUnkClip)
```

```
003596                  ELSE
003597                      BEGIN
003598                      clipboard.Bind;
003599                  {$H-} GetGrScrap(pic); {$H+}
003600
003601                      window := clipboard.window;
003602                      IF window = NIL THEN
003603                          SELF.DoPaste(NIL, pic, cmdPhase)
003604                      ELSE
003605                          BEGIN
003606                          selection := window.selectPanel.selection;
003607                          IF selection.Class = cSelection THEN
003608                              SELF.DoPaste(NIL, pic, cmdPhase)
003609                          ELSE
003610                              SELF.DoPaste(selection, pic, cmdPhase);
003611                          END;
003612
003613                      clipboard.Unbind;
003614                      END;
003615              undoPhase:
003616                  SELF.DoPaste(NIL, NIL, cmdPhase);
003617              END;
003618          {$IFC fTrace}EP;{$ENDC}
003619      END;
003620
003621
003622  {$S SgABCini}
003623  END;
003624  {$S SgABCres}

End of File -- Lines: 3624 Characters: 116410
```

```
================================================================================================
FILE: "LIBTK/UABC3.TEXT"
================================================================================================

000001  {INCLUDE FILE UABC3 -- IMPLEMENTATION OF UABC}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004          {TImage-TView-TPaginatedView-TPageView-TPrintManager-THeading-TSelection}
000005
000006
000007
000008  METHODS OF TImage;
000009
000010
000011      {$S SgABCini}
000012      FUNCTION  {TImage.}CREATE{(object: TObject; heap: THeap; itsExtent: LRect; itsView: TView): TImage};
000013      BEGIN
000014          {$IFC fTrace}BP(7);{$ENDC}
000015          IF object = NIL THEN
000016              object := NewObject(heap, THISCLASS);
000017          SELF := TImage(object);
000018
000019          WITH SELF DO
000020              BEGIN
000021              extentLRect := itsExtent;
000022              view := itsView;
000023              allowMouseOutside := FALSE;
000024              END;
000025          {$IFC fTrace}EP;{$ENDC}
000026      END;
000027      {$S SgABCres}
000028
000029
000030      {$IFC fDebugMethods}
000031      {$S SgABCdbg}
000032      PROCEDURE {TImage.}Fields{(PROCEDURE Field(nameAndType: S255))};
000033      BEGIN
000034          Field('extentLRect: LRect');
000035          Field('view: TView');
000036          Field('allowMouseOutside: BOOLEAN');
000037          Field('');
000038      END;
000039      {$S SgABCres}
000040      {$ENDC}
000041
000042
000043      {$S Override}
```

```
000044      FUNCTION {TImage.}CursorAt{(mouseLPt: LPoint): TCursor};
000045      BEGIN
000046          {$IFC fTrace}BP(3);{$ENDC}
000047          CursorAt := NoCursor;
000048          {$IFC fTrace}EP;{$ENDC}
000049      END;
000050
000051
000052      {$S Override}
000053      PROCEDURE {TImage.}Draw;
000054      BEGIN
000055          {$IFC fTrace}BP(3);{$ENDC}
000056          {$IFC fTrace}EP;{$ENDC}
000057      END;
000058
000059
000060      {$S Override}
000061      PROCEDURE {TImage.}EachActualPart{(PROCEDURE DoToObject(filteredObj: TObject))};
000062      BEGIN
000063          {$IFC fTrace}BP(3);{$ENDC}
000064          SELF.view.panel.window.EachActualPart(DoToObject);
000065          {$IFC fTrace}EP;{$ENDC}
000066      END;
000067
000068
000069      {$S sFilter}
000070      PROCEDURE {TImage.}EachVirtualPart{(PROCEDURE DoToObject(filteredObj: TObject))};
000071      BEGIN
000072          {$IFC fTrace}BP(3);{$ENDC}
000073          SELF.view.panel.window.FilterDispatch(NIL, SELF, DoToObject);
000074          {$IFC fTrace}EP;{$ENDC}
000075      END;
000076
000077
000078      {$S sFilter}
000079      PROCEDURE {TImage.}FilterAndDo{(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject))};
000080      BEGIN
000081          {$IFC fTrace}BP(11);{$ENDC}
000082          SELF.view.panel.window.FilterDispatch(actualObj, SELF, DoToObject);
000083          {$IFC fTrace}EP;{$ENDC}
000084      END;
000085
000086
000087      {$S sCldInit}
000088      PROCEDURE {TImage.}HaveView{(view: TView)};
000089      BEGIN
000090          {$IFC fTrace}BP(7);{$ENDC}
000091          SELF.view := view;  {fancier subclasses do fancier things here}
```

```
000092          {$IFC fTrace}EP;{$ENDC}
000093      END;
000094
000095
000096      {$S sRes}
000097      FUNCTION {TImage.}Hit{(mouseLPt: LPoint): BOOLEAN};
000098      BEGIN
000099          {$IFC fTrace}BP(3);{$ENDC}
000100          Hit := LRectHasLPt(SELF.extentLRect, mouseLPt);
000101          {$IFC fTrace}EP;{$ENDC}
000102      END;
000103
000104
000105      {$S sRes}
000106      PROCEDURE {TImage.}Invalidate;
000107      BEGIN
000108          {$IFC fTrace}BP(3);{$ENDC}
000109          IF thePad <> NIL THEN
000110              thePad.InvalLRect(SELF.extentLRect);
000111          {$IFC fTrace}EP;{$ENDC}
000112      END;
000113
000114
000115  {$S Override}
000116      FUNCTION {TImage.}LaunchLayoutBox{(view: TView): TImage};
000117      BEGIN
000118          {$IFC fTrace}BP(3);{$ENDC}
000119          LaunchLayoutBox := NIL;
000120          {$IFC fTrace}EP;{$ENDC}
000121      END;
000122
000123
000124      {$S sRes}
000125      PROCEDURE {TImage.}OffSetBy{(deltaLPt: LPoint)};
000126      BEGIN
000127          {$IFC fTrace}BP(3);{$ENDC}
000128    {$H-} OffsetLRect(SELF.extentLRect, deltaLPt.h, deltaLPt.v); {$H+}
000129          {$IFC fTrace}EP;{$ENDC}
000130      END;
000131
000132
000133      {$S sRes}
000134      PROCEDURE {TImage.}OffSetTo{(newTopLeft: LPoint)};
000135          VAR deltaLPt:   LPoint;
000136              curTopLeft: LPoint;
000137      BEGIN
000138          {$IFC fTrace}BP(3);{$ENDC}
000139          curTopLeft := SELF.extentLRect.topLeft;
```

```
000140          SetLPt(deltaLPt, newTopLeft.h - curTopLeft.h, newTopLeft.v - curTopLeft.v);
000141          SELF.OffsetBy(deltaLPt);
000142          {$IFC fTrace}EP;{$ENDC}
000143      END;
000144
000145
000146      {$S sRes}
000147      PROCEDURE {TImage.}MouseMove{(mouseLPt: LPoint)};
000148      BEGIN
000149          {$IFC fTrace}BP(7);{$ENDC}
000150          IF SELF.view.panel <> NIL THEN
000151              SELF.view.panel.selection.MouseMove(mouseLPt);
000152          {$IFC fTrace}EP;{$ENDC}
000153      END;
000154
000155
000156      {$S sRes}
000157      PROCEDURE {TImage.}MousePress{(mouseLPt: LPoint)};
000158      BEGIN
000159          {$IFC fTrace}BP(7);{$ENDC}
000160          IF SELF.view.panel <> NIL THEN
000161              SELF.view.panel.selection.MousePress(mouseLPt);
000162          {$IFC fTrace}EP;{$ENDC}
000163      END;
000164
000165
000166      {$S sRes}
000167      PROCEDURE {TImage.}MouseRelease;
000168      BEGIN
000169          {$IFC fTrace}BP(7);{$ENDC}
000170          IF SELF.view.panel <> NIL THEN
000171              SELF.view.panel.selection.MouseRelease;
000172          {$IFC fTrace}EP;{$ENDC}
000173      END;
000174
000175
000176      {$S sRes}
000177      PROCEDURE {TImage.}MouseTrack{(mPhase: TMousePhase; mouseLPt: LPoint)};
000178          VAR panel:  TPanel;
000179              window: TWindow;
000180      BEGIN
000181          {$IFC fTrace}BP(7);{$ENDC}
000182          panel := SELF.view.panel;
000183          IF panel <> NIL THEN
000184              BEGIN
000185              IF NOT (panel.selection.canCrossPanels OR SELF.allowMouseOutside) THEN
000186                  LRectHaveLPt(SELF.extentLRect, mouseLPt);
000187              window := panel.window;
```

```
000188              window.clickPanel := panel;
000189              END;
000190
000191          SELF.view.clickLPt := mouseLPt; {e.g., for Set Page Breaks use}
000192          CASE mPhase OF
000193              mPress:     SELF.MousePress(mouseLPt);
000194              mMove:      SELF.MouseMove(mouseLPt);
000195              mRelease:   BEGIN
000196                          SELF.MouseMove(mouseLPt);
000197                          window.Update(TRUE);
000198                          SELF.MouseRelease;
000199                          END;
000200              END;
000201          window.Update(TRUE);
000202          {$IFC fTrace}EP;{$ENDC}
000203      END;
000204
000205
000206      {$S Override}
000207      PROCEDURE {TImage.}ReactToPrinterChange;
000208      BEGIN
000209          {$IFC fTrace}BP(3);{$ENDC}
000210          {$IFC fTrace}EP;{$ENDC}
000211      END;
000212
000213
000214      {$S Override}
000215      PROCEDURE {TImage.}RecalcExtent;
000216      BEGIN
000217          {$IFC fTrace}BP(3);{$ENDC}
000218          {$IFC fTrace}EP;{$ENDC}
000219      END;
000220
000221
000222      {$S sRes}
000223      PROCEDURE {TImage.}Resize{(newExtent: LRect)};
000224      BEGIN
000225          {$IFC fTrace}BP(3);{$ENDC}
000226          SELF.extentLRect := newExtent;
000227          {$IFC fTrace}EP;{$ENDC}
000228      END;
000229
000230
000231      {$S sRes}
000232      FUNCTION {TImage.}SeesSameAs{(image: TImage): BOOLEAN; DEFAULT};    {$}
000233      BEGIN
000234          {$IFC fTrace}BP(3);{$ENDC}
000235          SeesSameAs := image = SELF;
```

```
000236          {$IFC fTrace}EP;{$ENDC}
000237      END;
000238
000239
000240  {$S SgABCini}
000241  END;
000242
000243
000244
000245  METHODS OF TView;
000246
000247
000248      {$S SgABCini}
000249      FUNCTION  {TView.}CREATE{(object: TObject; heap: THeap; itsPanel: TPanel; itsExtent: LRect;
000250                  itsPrintManager: TPrintManager; itsDfltMargins: LRect; itsFitPagesPerfectly:BOOLEAN;
000251                  itsRes: Point; isMainView: BOOLEAN): TView};
000252              VAR screenPad:  TPad;
000253      BEGIN
000254          {$IFC fTrace}BP(7);{$ENDC}
000255          IF object = NIL THEN
000256              object := NewObject(heap, THISCLASS);
000257          SELF := TView(TImage.CREATE(object, heap, itsExtent, NIL));
000258
000259          WITH SELF DO
000260              BEGIN
000261              view := SELF;
000262              panel := itsPanel;
000263              printManager := itsPrintManager;
000264              res := itsRes;
000265              clickLPt := itsExtent.topLeft;
000266              fitPagesPerfectly := itsFitPagesPerfectly;
000267              {$H-}
000268              SetPt(scrollPastEnd, 60, 40);
000269              {$H+}
000270              END;
000271          SELF.isMainView := isMainView;
000272          SELF.isPrintable := (itsPrintManager <> NIL) AND isMainView;
000273
000274          screenPad := TPad.CREATE(NIL, heap, zeroRect, zeroLRect, screenRes, SELF.res, NIL);
000275          SELF.screenPad := screenPad;
000276
000277      {$H-}SetLPt(SELF.stdScroll, (16 * SELF.res.h) DIV screenRes.h, (11 * SELF.res.v) DIV screenRes.v); {$H+}
000278
000279          IF isMainView THEN
000280              BEGIN
000281              itsPanel.HaveView(SELF);
000282              IF itsPrintmanager <> NIL THEN
000283                  itsPrintManager.Init(SELF, itsDfltMargins);
```

```
000284              SELF.ReactToPrinterChange;
000285              END;
000286          {$IFC fTrace}EP;{$ENDC}
000287      END;
000288      {$S SgABCres}
000289
000290
000291
000292      {$S SgABCini}
000293      PROCEDURE {TView.}Free;
000294      BEGIN
000295          {$IFC fTrace}BP(7);{$ENDC}
000296          IF SELF.isMainView THEN
000297              Free(SELF.printManager);
000298          Free(SELF.screenPad);
000299          SUPERSELF.Free;
000300          {$IFC fTrace}EP;{$ENDC}
000301      END;
000302      {$S SgABCres}
000303
000304
000305      {$IFC fDebugMethods}
000306      {$S SgABCdbg}
000307      PROCEDURE {TView.}Fields{(PROCEDURE Field(nameAndType: S255))};
000308      BEGIN
000309          TImage.Fields(Field);
000310          Field('panel: TPanel');
000311          Field('clickLPt: LPoint');
000312          Field('printManager: TPrintManager');
000313          Field('res: Point');
000314          Field('screenPad: TPad');
000315          Field('fitPagesPerfectly: BOOLEAN');
000316          Field('isPrintable: BOOLEAN');
000317          Field('isMainView: BOOLEAN');
000318          Field('stdScroll: LPoint');
000319          Field('scrollPastEnd: Point');
000320          Field('');
000321      END;
000322      {$S SgABCres}
000323      {$ENDC}
000324
000325
000326      {$S SgABCpri}
000327      PROCEDURE {TView.}AddStripOfPages{(vhs: VHSelect)};
000328      BEGIN
000329          {$IFC fTrace}BP(7);{$ENDC}
000330          IF SELF.printManager <> NIL THEN
000331              SELF.printManager.AddStripOfPages(vhs);
```

```
000332          {$IFC fTrace}EP;{$ENDC}
000333      END;
000334      {$S SgABCres}
000335
000336
000337     {$S sCldInit}
000338      PROCEDURE {TView.}BeInPanel{(panel: TPanel)};
000339      BEGIN
000340          {$IFC fTrace}BP(7);{$ENDC}
000341          SELF.panel := panel;
000342          {$IFC fTrace}EP;{$ENDC}
000343      END;
000344      {$S SgABCres}
000345
000346
000347      {$S SgABCcld}
000348      PROCEDURE {TView.}CreateUniversalText;
000349      BEGIN
000350          {$IFC fTrace}BP(6);{$ENDC}
000351          {$IFC fTrace}EP;{$ENDC}
000352      END;
000353      {$S SgABCres}
000354
000355
000356      {$S sRes}
000357      FUNCTION  {TView.}CursorAt{(mouseLPt: LPoint): TCursorNumber};
000358      BEGIN
000359          {$IFC fTrace}BP(2);{$ENDC}
000360          CursorAt := arrowCursor;
000361          {$IFC fTrace}EP;{$ENDC}
000362      END;
000363
000364
000365  {$S SgDRWres}
000366      FUNCTION  {TView.}DoReceive{(selection: TSelection; lPtInView: LPoint): BOOLEAN};
000367      BEGIN
000368          {$IFC fTrace}BP(11);{$ENDC}
000369          DoReceive := FALSE; {Default is to refuse cross-panel drag}
000370          {$IFC fTrace}EP;{$ENDC}
000371      END;
000372  {$S SgABCres}
000373
000374
000375      {$S sStartup}
000376      FUNCTION  {TView.}ForceBreakAt{(vhs: VHSelect; precedingLocation: LONGINT;
000377                                 proposedLocation: LONGINT): LONGINT};
000378      BEGIN
000379          {$IFC fTrace}BP(11);{$ENDC}
```

```
000380          ForceBreakAt := proposedLocation;  {default is to accept the proposal; client can override}
000381          {$IFC fTrace}EP;{$ENDC}
000382      END;
000383
000384
000385      {$S sScroll}
000386      PROCEDURE {TView.}GetStdScroll{(VAR deltaLStd: LPoint)};
000387      BEGIN
000388          {$IFC fTrace}BP(3);{$ENDC}
000389          IF NOT SELF.panel.zoomed THEN
000390              deltaLStd := SELF.stdScroll
000391          ELSE
000392              WITH SELF.panel.zoomFactor DO
000393            {$H-} BEGIN
000394                  deltaLStd.h := LIntOvrInt(LIntMulInt(ORD4(SELF.stdScroll.h), denominator.h), numerator.h);
000395                  deltaLStd.v := LIntOvrInt(LIntMulInt(ORD4(SELF.stdScroll.v), denominator.v), numerator.v);
000396        {$H+} END;
000397          {$IFC fTrace}EP;{$ENDC}
000398      END;
000399
000400
000401      {$S SgABCpri}
000402      FUNCTION  {TView.}MaxPageToPrint{: LONGINT};
000403      BEGIN
000404          {$IFC fTrace}BP(7);{$ENDC}
000405          MaxPageToPrint := SELF.printManager.breaks[v].size * SELF.printManager.breaks[h].size;
000406          {$IFC fTrace}EP;{$ENDC}
000407      END;
000408      {$S SgABCres}
000409
000410
000411      {$S sStartup}
000412      FUNCTION  {TView.}NoSelection{: TSelection};
000413      BEGIN
000414          {$IFC fTrace}BP(6);{$ENDC}
000415          NoSelection := TSelection.CREATE(NIL, SELF.Heap, SELF, nothingKind, zeroLPt);
000416          {$IFC fTrace}EP;{$ENDC}
000417      END;
000418
000419
000420      {$S sRes}
000421      FUNCTION  {TView.}OKToDrawIn{(lRectInView: LRect): BOOLEAN};
000422      BEGIN
000423          {$IFC fTrace}BP(6);{$ENDC}
000424          OKToDrawIn := FALSE;   {The default is to assume the worst, unless the application overrides}
000425          {$IFC fTrace}EP;{$ENDC}
000426      END;
000427
```

```
000428
000429      {$S sCldInit}
000430      PROCEDURE {TView.}ReactToPrinterChange;
000431      BEGIN
000432          {$IFC fTrace}BP(7);{$ENDC}
000433          IF SELF.printManager <> NIL THEN
000434              SELF.printManager.ReactToPrinterChange;
000435          {$IFC fTrace}EP;{$ENDC}
000436      END;
000437      {$S SgABCres}
000438
000439
000440      {$S sStartup}
000441      PROCEDURE {TView.}RedoBreaks;
000442      BEGIN
000443          {$IFC fTrace}BP(6);{$ENDC}
000444          IF SELF.printManager <> NIL THEN
000445              SELF.printManager.RedoBreaks;
000446          {$IFC fTrace}EP;{$ENDC}
000447      END;
000448
000449
000450      {$S SgABCcld}
000451      PROCEDURE {TView.}RemapManualBreaks{(
000452              FUNCTION NewBreakLocation(vhs: VHSelect; oldBreak: LONGINT): LONGINT)};
000453          VAR printManager:   TPrintManager;
000454              oldLoc:         LONGINT;
000455              newLoc:         LONGINT;
000456              oldIndex:       LONGINT;
000457              vhs:            VHSelect;
000458      BEGIN
000459          {$IFC fTrace}BP(6);{$ENDC}
000460          printManager := SELF.printManager;
000461          IF printManager <> NIL THEN
000462              BEGIN
000463              printManager.ClearPageBreaks(TRUE);
000464              FOR vhs := v TO h DO
000465                  FOR oldIndex := 1 TO printManager.breaks[vhs].size - 1 DO
000466                  BEGIN
000467                  oldLoc := TpLONGINT(printManager.breaks[vhs].At(oldIndex))^;
000468                  newLoc := - NewBreakLocation(vhs, ABS(oldLoc));
000469                  printManager.breaks[vhs].PutAt(oldIndex, @newLoc);
000470                  END;
000471              SELF.RedoBreaks;
000472              END;
000473          {$IFC fTrace}EP;{$ENDC}
000474      END;
000475
```

```
000476
000477      {$S sCldInit}
000478      PROCEDURE {TView.}Resize{(newExtent: LRect)};
000479          VAR s:              TListScanner;
000480              pageBreak:      LONGINT;
000481              vhs:            VHSelect;
000482              oldLimit:       LONGINT;
000483              newLimit:       LONGINT;
000484              breakIndex:     INTEGER;
000485              breakArray:     TArray;
000486      BEGIN
000487          {$IFC fTrace}BP(9);{$ENDC}
000488          IF NOT (SELF.isMainView) OR NOT(SELF.isPrintable) THEN
000489              SUPERSELF.Resize(newExtent)
000490          ELSE
000491              IF NOT EqualLRect(SELF.extentLRect, newExtent) THEN
000492                  BEGIN
000493                  FOR vhs := v TO h DO
000494                      BEGIN
000495                      oldLimit := SELF.extentLRect.botRight.vh[orthogonal[vhs]];
000496                      newLimit := newExtent.botRight.vh[orthogonal[vhs]];
000497
000498                      breakIndex := 1;
000499                      breakArray := SELF.printManager.breaks[vhs];
000500
000501                      WHILE breakIndex <= breakArray.size DO
000502                          BEGIN
000503                          pageBreak := TpLONGINT(breakArray.At(breakIndex))^;
000504                          IF pageBreak = oldLimit THEN
000505                              BEGIN
000506                              {reset the end-of-view pagebreak to new limit}
000507                              pageBreak := newLimit;
000508                              breakArray.PutAt(breakIndex, @pageBreak);
000509                              END
000510                          ELSE
000511                          IF ABS(pageBreak) >= newLimit THEN
000512                              {discard other now-too-big pagebreaks}
000513                              BEGIN
000514                              breakArray.DelAt(breakIndex);
000515                              breakIndex := breakIndex - 1;
000516                              END;
000517                          breakIndex := breakIndex + 1;
000518                          {ELSE pagebreak still valid; do nothing}
000519                          END;
000520                      END;
000521                  SELF.extentLRect := newExtent;
000522                  SELF.panel.Rescroll;
000523                  END;
```

```
000524          {$IFC fTrace}EP;{$ENDC}
000525      END;
000526      {$S SgABCres}
000527
000528
000529      {$S SgABCcld}
000530      PROCEDURE {TView.}SetFunctionValue{(keyword: S255; VAR itsValue: S255)};
000531      BEGIN
000532          {$IFC fTrace}BP(6);{$ENDC}
000533          StrUpperCased(@keyword);
000534
000535          IF keyword = varPage THEN  {+SW+}
000536              LIntToStr(theMarginPad.pageNumber, @itsValue)
000537          ELSE
000538          IF keyword = varTitle THEN {+SW+}
000539              SELF.panel.window.GetTitle(itsValue)
000540
000541         {ELSE
000542          IF keyword = ....   *** this is where to add more predefined functions ***}
000543
000544          ELSE {didn't parse}
000545              itsValue := keyword;
000546          {$IFC fTrace}EP;{$ENDC}
000547      END;
000548
000549
000550      {$S sStartup}
000551      PROCEDURE {TView.}SetMinViewSize{(VAR minLRect: LRect)};
000552      BEGIN
000553          {$IFC fTrace}BP(6);{$ENDC}
000554          minLRect := SELF.extentLRect; {client may override this to inspect his view for other ideas}
000555          {$IFC fTrace}EP;{$ENDC}
000556      END;
000557
000558
000559  {$S SgABCini}
000560  END;
000561  {$S SgABCres}
000562
000563
000564
000565  METHODS OF TPaginatedView;
000566
000567
000568  {$S SgABCpri}
000569      FUNCTION {TPaginatedView.}CREATE{(object: TObject; heap: THeap; itsUnpaginatedView: TView)
000570                                       : TPaginatedView};
000571          VAR viewExtent:         LRect;
```

```
000572              pgsPerRowStrip:     INTEGER;
000573              pgsPerColStrip:     INTEGER;
000574              pageWidth:          LONGINT;
000575              pageHeight:         LONGINT;
000576              printerMetrics:     TPrinterMetrics;
000577              pageList:           TList;
000578              rowStrip:           INTEGER;
000579              colStrip:           INTEGER;
000580              pageOrigin:         LPoint;
000581      BEGIN
000582          {$IFC fTrace}BP(7);{$ENDC}
000583          printerMetrics := itsUnpaginatedView.printManager.printerMetrics;
000584          pgsPerRowStrip := itsUnpaginatedView.printManager.breaks[v].size;
000585          pgsPerColStrip := itsUnpaginatedView.printManager.breaks[h].size;
000586
000587          WITH printerMetrics, paperRect DO
000588              BEGIN
000589              pageWidth := LIntOvrInt(LIntMulInt(ORD4(right - left), itsUnpaginatedView.res.h),
000590                                      printerMetrics.res.h);
000591              pageHeight:= LIntOvrInt(LIntMulInt(ORD4(bottom - top), itsUnpaginatedView.res.v),
000592                                      printerMetrics.res.v);
000593              END;
000594
000595          SetLRect(viewExtent, 0, 0, pgsPerRowStrip * ORD4(pageWidth), pgsPerColStrip * ORD4(pageHeight));
000596
000597          IF object = NIL THEN
000598              object := NewObject(heap, THISCLASS);
000599          SELF := TPaginatedView(TView.CREATE(object, heap, itsUnpaginatedView.panel, viewExtent,
000600                                          itsUnpaginatedView.printManager, zeroLRect, FALSE,
000601                                          itsUnpaginatedView.res, FALSE));
000602
000603          WITH SELF DO
000604              BEGIN
000605              unpaginatedView := itsUnpaginatedView;
000606              pageSize[h] := pageWidth;
000607              pageSize[v] := pageHeight;
000608              workingInMargins := FALSE;
000609              END;
000610
000611         {$IFC fTrace}EP;{$ENDC}
000612      END;
000613
000614
000615      {$IFC fDebugMethods}
000616      {$S SgABCdbg}
000617      PROCEDURE {TPaginatedView.}Fields{(PROCEDURE Field(nameAndType: S255))};
000618      BEGIN
000619          TView.Fields(Field);
```

```
000620          Field('unPaginatedView: TView');
000621          Field('pageSize: ARRAY[0..1] OF LONGINT');
000622          Field('workingInMargins: BOOLEAN');
000623          Field('');
000624      END;
000625      {$S SgABCcld}
000626      {$ENDC}
000627
000628
000629      {$S SgABCpri}
000630      PROCEDURE {TPaginatedView.}AddStripOfPages{(vhs: VHSelect)};
000631          VAR panel:  TPanel;
000632      BEGIN
000633          {$IFC fTrace}BP(9);{$ENDC}
000634          panel := SELF.panel;
000635          panel.Preview(mPrvwOff); {get back to main-view metrics in the panes}
000636         {Don't refer to SELF after this, since Preview has deallocated me}
000637          panel.view.printManager.AddStripOfPages(vhs);
000638          panel.Preview(mPrvwMargins); {creates fresh paginated view with correct info}
000639          {$IFC fTrace}EP;{$ENDC}
000640      END;
000641
000642
000643      {$S SgABCpri}
000644      PROCEDURE {TPaginatedView.}AdornPageOnScreen;  {+SW+} {now using CONSTs to tune layout of pg numbers}
000645          CONST
000646  {$IFC LibraryVersion <= 20}
000647                  lrOffset = 10;
000648                  topOffset = 22;
000649                  bottomOffset = 9;
000650
000651                  lrOutset = 6;
000652                  topOutset = 4;
000653                  bottomOutset = 2;
000654  {$ELSEC}
000655                  lrOffset = 10;
000656                  topOffset = 22;
000657                  bottomOffset = 9;
000658
000659                  lrOutset = 6;
000660                  topOutset = 1;  {+SW+}
000661                  bottomOutset = 1;
000662  {$ENDC}
000663          VAR pgNum:          S255;
000664              r:              Rect;
000665              tempRect:       Rect;
000666              paperRect:      Rect;
000667              contentRect:    Rect;
```

```
000668              pat:            pattern;
000669  {$IFC LibraryVersion <= 20}
000670              fInfo:          TFInfo;
000671  {$ELSEC}
000672              fInfo:          FontInfo;
000673  {$ENDC}
000674              numberLength:   INTEGER;
000675              printManager:   TPrintManager;
000676
000677          PROCEDURE DistinguishScreenFeedback(theString: S255; h, v: INTEGER);
000678                  VAR box: Rect;
000679              BEGIN
000680              WITH box, fInfo DO
000681                  BEGIN
000682                  left := h - lrOutset;
000683                  right := h + StringWidth(theString) + lrOutset;
000684                  top := v - ascent - leading - topOutset;
000685                  bottom := v + descent + leading + bottomOutset;
000686                  END;
000687              FillRoundRect(box, 10, 10, white);{*** These constants won't stand up under zooming! ***}
000688              MoveTo(h, v);
000689              DrawString(theString);
000690              InvertRoundRect(box, 10, 10);
000691              END;
000692      BEGIN
000693          {$IFC fTrace}BP(7);{$ENDC}
000694          printManager := SELF.unpaginatedView.printManager;
000695
000696          IF focusArea <> theMarginPad THEN   {need to refocus onto the exterior...}
000697              theMarginPad.Focus;
000698
000699          {frame the overall page}
000700          penNormal;
000701          penMode(patOr);
000702          penSize(3,2);
000703          FrameLRect(printManager.paperLRect);
000704
000705          {draw a very light-gray pattern everywhere in the margins}
000706          theMarginPad.LRectToRect(printManager.paperLRect, paperRect);
000707          RectRgn(padRgn, paperRect);
000708          theMarginPad.LRectToRect(printManager.contentLRect, contentRect);
000709          RectRgn(scrRgn1ForDrawHdgs, contentRect);
000710          DiffRgn(padRgn, scrRgn1ForDrawHdgs, scrRgn1ForDrawHdgs);
000711          PenMode(patOr);
000712          theMarginPad.LPatToPat(marginPattern, pat);
000713          PenPat(pat);
000714          PaintRgn(scrRgn1ForDrawHdgs);
000715
```

```
000716          IF NOT EqualRect(theBodyPad.nonNullBody, theBodyPad.innerRect) THEN
000717              BEGIN
000718              RectRgn(scrRgn1ForDrawHdgs, theBodyPad.innerRect);
000719              RectRgn(scrRgn2ForDrawHdgs, theBodyPad.nonNullBody);
000720              DiffRgn(scrRgn1ForDrawHdgs, scrRgn2ForDrawHdgs, scrRgn1ForDrawHdgs);
000721
000722              {Both theBodyPad.innerRect & theBodyPad.nonNullBody are expressed in (0,0)-origined
000723                  window coordinates; since we are focused on theMarginPad now, must offset the
000724                  rgn by its origin.}
000725              WITH theMarginPad.origin DO
000726                  {$H-}
000727                  OffsetRgn(scrRgn1ForDrawHdgs, h, v);
000728                  {$H+}
000729
000730              thePad.SetPen(limboPen);
000731              PaintRgn(scrRgn1ForDrawHdgs);
000732              END;
000733
000734        {Frame the content rectangle--normally directly abuts the margin}
000735          penNormal;
000736          penMode(patOr);
000737          FrameRect(contentRect);
000738
000739        {draw page numbers in corners}
000740          IntToStr(theMarginPad.pageNumber, @pgNum);
000741          SetQDTypeStyle(cornerNumberStyle);
000742          numberLength := StringWidth(pgNum);
000743          r := paperRect;
000744          GetFontInfo(fInfo);
000745          DistinguishScreenFeedback(pgNum, r.left + lrOffset, r.top + topOffset);
000746          DistinguishScreenFeedback(pgNum, r.right - numberLength - lrOffset, r.top + topOffset);
000747          DistinguishScreenFeedback(pgNum, r.right - numberLength - lrOffset, r.bottom - bottomOffset);
000748          DistinguishScreenFeedback(pgNum, r.left + lrOffset, r.bottom - bottomOffset);
000749
000750          {$IFC fTrace}EP;{$ENDC}
000751      END;
000752      {$S SgABCres}
000753
000754
000755
000756     {$S SgABCpri}
000757     FUNCTION  {TPaginatedView.}CursorAt{(mouseLPt: LPoint): TCursorNumber};
000758                                             {later deal with cursor for margins}
000759          VAR unPagLPt: LPoint;
000760      BEGIN
000761         {$IFC fTrace}BP(9);{$ENDC}
000762         SELF.DepagifyLPoint(mouseLPt, unPagLPt);
000763         CursorAt := SELF.unpaginatedView.CursorAt(unPagLPt);
```

```
000764          {$IFC fTrace}EP;{$ENDC}
000765      END;
000766
000767
000768    PROCEDURE {TPaginatedView.}DepagifyLPoint{(pagLPt: LPoint; VAR unPagLPt: LPoint)};
000769     {Given a point in the paginated view, determine the nearest corresponding point in the unpaginated view}
000770
000771          VAR printManager:   TPrintManager;
000772              meatLRect:      LRect;      {the portion of the page that displays a part of the main view}
000773              vhs:            VHSelect;
000774              breakArray:     TArray {OF LONGINT};
000775              strip:          INTEGER;    {the ordinal number of the strip containing the page}
000776              breakLocation:  LONGINT;    {the coordinate of the start of the page}
000777              pageBreak:      LONGINT; {the page break at the beginning of the page}
000778              nextBreak:      LONGINT; {the page break at the end of the page}
000779              pageOrigin:     LPoint;     {the top left corner of the page, in the paginated view}
000780              strips:         Point;      {the strip numbers in each direction, stored as a Point}
000781              lOffsetPt:      LPoint;     {the top left corner of the meat rect of the page, in the main view}
000782
000783      BEGIN
000784          {$IFC fTrace}BP(7);{$ENDC}
000785          LRectHaveLPt(SELF.extentLRect, pagLPt);
000786
000787          printManager := SELF.unpaginatedView.printManager;
000788          meatLRect := printManager.contentLRect;
000789
000790          FOR vhs := v TO h DO
000791              BEGIN
000792              breakArray := printManager.breaks[orthogonal[vhs]];
000793
000794            {compute strip number}
000795              strip := Min(LIntDivLInt(pagLPt.vh[vhs], SELF.pageSize[vhs]) + 1, breakArray.size);
000796
000797            {compute breakLocation, being the location in the main view of the top-leftmost
000798                      content point of the page in which our boy was found}
000799              IF strip = 1 THEN
000800                  breakLocation := 0
000801              ELSE
000802                  BEGIN
000803                  pageBreak := TpLONGINT(breakArray.At(strip - 1))^;
000804                  breakLocation := ABS(pageBreak);
000805                  END;
000806
000807            {recompute end of meatLRect (limbo boundary)}
000808              nextBreak := TpLONGINT(breakArray.At(strip))^;
000809              meatLRect.botRight.vh[vhs] := meatLRect.topLeft.vh[vhs] + ABS(nextBreak) - breakLocation;
000810
000811            {compute pageOrigin -- the location in the paginated view of the topleft corner of this page}
```

```
000812              pageOrigin.vh[vhs] := LIntMulInt(SELF.pageSize[vhs], strip - 1);
000813
000814            {stuff strip and breakLocation into points for future reference}
000815              strips.vh[vhs] := strip;
000816              lOffsetPt.vh[vhs] := breakLocation;
000817              END;
000818
000819          {project the point into the (0,0)-origined space that the printManager rectangles are in}
000820            LPtMinusLPt(pagLPt, pageOrigin, pagLPt);
000821            LRectHaveLPt(meatLRect, pagLPt);  {force it to meat rectangle}
000822            LPtMinusLPt(pagLPt, meatLRect.topLeft, pagLPt); {get offset from inner corner}
000823            LPtPlusLPt(pagLPt, lOffsetPt, unPagLPt);  {project onto main view}
000824            {$IFC fTrace}EP;{$ENDC}
000825        END;
000826
000827
000828        PROCEDURE {TPaginatedView.}DoOnPages{(focusOnInterior: BOOLEAN; PROCEDURE DoOnAPage))};
000829            VAR pgsPerStrip:    INTEGER;  {pages per row-strip if pageRiseDirection = h}
000830                firstRowStrip:  INTEGER;
000831                firstColStrip:  INTEGER;
000832                lastRowStrip:   INTEGER;
000833                lastColStrip:   INTEGER;
000834                row:            INTEGER;
000835                column:         INTEGER;
000836                pageNumber:     LONGINT;
000837                lOrigin:        LPoint;
000838                origin:         Point;
000839                anLRect:        LRect;
000840                incomingPane:   TPane;
000841        BEGIN
000842            {$IFC fTrace}BP(7);{$ENDC}
000843                incomingPane := TPane(thePad);
000844                anLRect := thePad.visLRect;
000845                IF SectLRect(anLRect, SELF.extentLRect, anLRect) THEN
000846                    {thanks for the lovely intersection};
000847
000848                IF EqualLRect(anLRect, zeroLRect) THEN
000849                    BEGIN
000850                    {$IFC fTrace}EP;{$ENDC}
000851                    EXIT(DoOnPages);
000852                    END;
000853
000854                pgsPerStrip := SELF.printManager.breaks[
000855                                    orthogonal[SELF.printManager.pageRiseDirection]].size;
000856
000857
000858                WITH anLRect, SELF DO
000859                    BEGIN
```

```
000860          {$H-}  firstRowStrip := LIntDivLInt(topLeft.v, pageSize[v]) + 1;
000861                 firstColStrip := LIntDivLInt(topLeft.h, pageSize[h]) + 1;
000862                 lastRowStrip :=  MIN(LIntDivLInt(botRight.v, pageSize[v]) + 1,
000863                         SELF.printManager.breaks[h].size);
000864                 lastColStrip :=  MIN(LIntDivLInt(botRight.h, pageSize[h]) + 1,
000865                         SELF.printManager.breaks[v].size);
000866          {$H+}  END;
000867
000868             PushFocus;
000869             IF (theMarginPad.view <> SELF.unpaginatedView) OR (theMarginPad.port = printerPseudoPort) THEN
000870                theMarginPad.Rework(SELF.unpaginatedView, zeroPt, screenRes, 1,
000871                         SELF.panel.zoomFactor, POINTER(SELF.panel.window.wmgrId));
000872
000873             FOR row := firstRowStrip TO lastRowStrip DO
000874                 FOR column := firstColStrip to lastColStrip DO
000875                     BEGIN
000876                     IF SELF.printManager.pageRiseDirection = h THEN
000877                         pageNumber := (row - 1) * pgsPerStrip + column
000878                     ELSE
000879                         pageNumber := (column - 1) * pgsPerStrip + row;
000880                     SetLPt(lOrigin,
000881                             LIntMulInt(SELF.pageSize[h], column - 1)  - incomingPane.scrollOffset.h,
000882                             LIntMulInt(SELF.pageSize[v], row - 1)     - incomingPane.scrollOffset.v);
000883                     SELF.screenPad.LPtToPt(lOrigin, origin);
000884
000885                     theMarginPad.SetForPage(pageNumber, origin);
000886
000887                     theMarginPad.ClipFurtherTo(incomingPane.innerRect); {clip page down to pane}
000888                     theBodyPad.ClipFurtherTo(incomingPane.innerRect);   {ditto page body}
000889                     IF focusOnInterior THEN
000890                         theBodyPad.Focus
000891                     ELSE
000892                         theMarginPad.Focus;
000893
000894                     DoOnAPage;
000895
000896                     END;
000897             PopFocus;
000898         {$IFC fTrace}EP;{$ENDC}
000899      END;
000900
000901
000902      PROCEDURE {TPaginatedView.}Draw;
000903          PROCEDURE DrawPageOnScreen;
000904          BEGIN
000905              SELF.printManager.DrawPage;
000906              SELF.AdornPageOnScreen;
000907          END;
```

```
000908      BEGIN
000909          {$IFC fTrace}BP(9);{$ENDC}
000910          SELF.DoOnPages(FALSE, DrawPageOnScreen);
000911          {$IFC fTrace}EP;{$ENDC}
000912      END;
000913
000914
000915      PROCEDURE {TPaginatedView.}MouseTrack{(mPhase: TPhase; mouseLPt: LPoint)};
000916          VAR unPagLPt: LPoint;
000917      BEGIN
000918          {$IFC fTrace}BP(9);{$ENDC}
000919          SELF.DepagifyLPoint(mouseLPt, unPagLPt);
000920          SELF.unpaginatedView.MouseTrack(mphase, unPagLPt);
000921          {$IFC fTrace}EP;{$ENDC}
000922      END;
000923
000924
000925      PROCEDURE {TPaginatedView.}PagifyLPoint{(unPagLPt: LPoint; VAR pagLPt: LPoint)};
000926          VAR pageBreak:  LONGINT;
000927              strip:      Point;
000928              vhs:        VHSelect;
000929              pageNumber: LONGINT;
000930              orthoVhs:   VHSelect;
000931      BEGIN
000932          {$IFC fTrace}BP(9);{$ENDC}
000933          pageNumber := SELF.printManager.PageWith(unPagLPt, strip);
000934          FOR vhs := v TO h DO
000935              BEGIN
000936              orthoVhs := orthogonal[vhs];
000937              IF (strip.vh[orthoVhs] < 1) OR (strip.vh[orthoVhs] > SELF.printManager.breaks[orthoVhs].Size) THEN
000938                  ABCBreak('PagifyLPt: strip=', strip.vh[orthoVHs])      {only for short-term debugging}
000939              ELSE
000940              IF strip.vh[orthoVhs] = 1 THEN
000941                  pagLPt.vh[vhs] := unPagLPt.vh[vhs] + SELF.printManager.contentLRect.topLeft.vh[vhs]
000942              ELSE
000943                  BEGIN
000944                  pageBreak := TpLONGINT(SELF.printManager.breaks[orthoVhs].At(strip.vh[orthoVhs] - 1))^;
000945                  pagLPt.vh[vhs] := unPagLPt.vh[vhs] + SELF.printManager.contentLRect.topLeft.vh[vhs]
000946                          + LIntMulInt(SELF.pageSize[vhs], strip.vh[orthoVhs] - 1) - ABS(pageBreak);
000947                  END;
000948              END;
000949          {$IFC fTrace}EP;{$ENDC}
000950      END;
000951
000952
000953
000954     {$S SgABCpri}
000955      PROCEDURE {TPaginatedView.}ReactToPrinterChange;
```

```
000956          VAR panel:  TPanel;
000957      BEGIN
000958          {$IFC fTrace}BP(9);{$ENDC}
000959          panel := SELF.panel;
000960          panel.Preview(mPrvwOff); {get back to main-view metrics in the panes}
000961         {Don't refer to SELF after this, since Preview has deallocated me}
000962          panel.view.ReactToPrinterChange;
000963          panel.Preview(mPrvwMargins); {creates fresh paginated view with correct info}
000964          {$IFC fTrace}EP;{$ENDC}
000965      END;
000966      {$S SgABCres}
000967
000968
000969      PROCEDURE {TPaginatedView.}RedoBreaks;
000970          VAR panel: TPanel;
000971      BEGIN
000972          {$IFC fTrace}BP(9);{$ENDC}
000973          panel := SELF.panel;
000974          panel.Preview(mPrvwOff); {get back to main-view metrics in the panes}
000975         {Don't refer to SELF after this, since Preview has deallocated me}
000976          panel.view.ReDoBreaks;
000977          panel.Preview(mPrvwMargins); {creates fresh paginated view with correct info}
000978          {$IFC fTrace}EP;{$ENDC}
000979      END;
000980
000981
000982  {$S SgABCini}
000983  END;
000984  {$S SgABCres}
000985
000986
000987  METHODS OF TPageView;
000988
000989  {$S sCldInit}
000990      FUNCTION  {TPageView.}CREATE{(object: TObject; heap: THeap; itsPrintManager: TPrintManager): TPageView};
000991          VAR view:       TView;
000992      BEGIN
000993         {$IFC fTrace}BP(9);{$ENDC}
000994          view := itsPrintManager.view;
000995          IF object = NIL THEN
000996              object := NewObject(heap, THISCLASS);
000997          SELF := TPageView(TView.CREATE(object, heap, view.panel, itsPrintManager.paperLRect,
000998                                  itsPrintManager, zeroLRect, FALSE, view.res, FALSE));
000999         {$IFC fTrace}EP;{$ENDC}
001000      END;
001001  {$S SgABCres}
001002
001003
```

```
001004  {$S SgABCpri}
001005      PROCEDURE  {TPageView.}Draw;
001006          VAR s:          TListScanner;
001007              heading:    THeading;
001008              pageNumber: LONGINT;
001009              outerFrame: LRect;
001010              headings:   TList;
001011              editing:    BOOLEAN;
001012      BEGIN
001013         {$IFC fTrace}BP(9);{$ENDC}
001014          PenNormal;
001015          IF SELF.printManager.frameBody THEN {body should be framed...}
001016              IF amPrinting THEN
001017                  FrameLRect(SELF.printManager.contentLRect);
001018
001019          editing := (SELF.printManager.layoutDialogBox <> NIL) AND
001020              (SELF.printManager.view.panel.window.dialogBox = SELF.printManager.layoutDialogBox);
001021
001022          headings := SELF.printManager.headings;
001023          IF headings <> NIL THEN
001024              BEGIN
001025              pageNumber := theMarginPad.pageNumber;
001026              s := headings.Scanner;   {tell each Heading to draw itself}
001027              WHILE s.Scan(heading) DO
001028                  IF heading.ShouldDraw(pageNumber)  THEN
001029                      BEGIN
001030                      IF NOT editing THEN
001031                          BEGIN
001032                          heading.AdjustForPage(pageNumber, FALSE);       {client changes contents/extent}
001033                          heading.LocateOnPage(FALSE);                    {...then we adjust to page}
001034                          END;
001035                      heading.Draw;
001036                      END;
001037              END;
001038         {$IFC fTrace}EP;{$ENDC}
001039      END;
001040  {$S SgABCres}
001041
001042
001043  {$S SgABCini}
001044  END;
001045  {$S SgABCres}
001046
001047
001048  {$S SgABCini}
001049  METHODS OF TPrintManager;
001050
001051
```

```
001052      FUNCTION  {TPrintManager.}CREATE{(object: TObject; heap: THeap): TPrintManager};
001053      BEGIN
001054          {$IFC fTrace}BP(6);{$ENDC}
001055          IF object = NIL THEN
001056              object := NewObject(heap, THISCLASS);
001057          SELF := TPrintManager(object);
001058          {$IFC fTrace}EP;{$ENDC}
001059      END;
001060
001061
001062      PROCEDURE  {TPrintManager.}Init{(itsMainView: TView; itsDfltMargins: LRect)};
001063          VAR paperLRect: LRect;
001064              l:          TArray;
001065              vhs:        VHSelect;
001066              pageView:   TView;
001067              pageBreak:  LONGINT;
001068              newList:    TList;
001069      BEGIN
001070          {$IFC fTrace}BP(6);{$ENDC}
001071          newList := TList.CREATE(NIL, itsMainView.Heap, 0);  {the Headings}
001072
001073          WITH SELF  DO
001074              BEGIN
001075              view := itsMainView;
001076              headings := newList;
001077              pageRiseDirection := h;
001078              frameBody := FALSE;
001079              layoutDialogBox := NIL;
001080              canEditPages := FALSE;     {subclass may make true}
001081              END;
001082
001083          FOR vhs := v TO h DO
001084              BEGIN
001085              l := TArray.CREATE(NIL, itsMainView.Heap, 1, SIZEOF(LONGINT));
001086              pageBreak := itsMainView.extentLRect.botRight.vh[orthogonal[vhs]];
001087              l.InsFirst(@pageBreak);
001088
001089              SELF.breaks[vhs] := l;
001090              END;
001091
001092          WITH itsDfltMargins DO
001093              BEGIN {$H-}
001094              left := ABS(left);
001095              top := ABS(top);
001096              right := - ABS(right);
001097              bottom := - ABS(bottom);
001098              END; {$H+}
001099          SELF.pageMargins := itsDfltMargins;
```

```
001100
001101          pageView := SELF.NewPageView(NIL);
001102          SELF.pageView := pageView;
001103
001104          SELF.SetDfltHeadings;        {NB: TView.CREATE will, after calling me, call ReactToPrinterChange;
001105                                            until that's done, things are not necessarily in synch}
001106          {$IFC fTrace}EP;{$ENDC}
001107      END;
001108
001109
001110      {$S SgABCini}
001111      PROCEDURE {TPrintManager.}Free;
001112          VAR vhs:    VHSelect;
001113      BEGIN
001114          {$IFC fTrace}BP(2);{$ENDC}
001115          FOR vhs := v TO h DO
001116              IF SELF.breaks[vhs] <> NIL THEN
001117                  SELF.breaks[vhs].Free;
001118          Free(SELF.pageView);
001119          SUPERSELF.Free;
001120          {$IFC fTrace}EP;{$ENDC}
001121      END;
001122      {$S SgABCres}
001123
001124
001125      {$IFC fDebugMethods}
001126      {$S SgABCdbg}
001127      PROCEDURE {TPrintManager.}Fields{(PROCEDURE Field(nameAndType: S255))};
001128      BEGIN
001129          Field('view: TView');
001130          Field('pageView: TPageView');
001131          Field('breaks: ARRAY[0..1] OF TArray');
001132          Field('pageMargins: LRect');
001133          Field('headings: TList');
001134          Field('canEditPages: BOOLEAN');
001135          Field('layoutDialogBox: TDialogBox');
001136          Field('frameBody: BOOLEAN');
001137          Field('paperLRect: LRect');
001138          Field('printableLRect: LRect');      {safeLRect out}
001139          Field('contentLRect: LRect');
001140          Field(CONCAT('printerMetrics: RECORD paperRect: Rect; printRect: Rect; ',
001141                       'res: Point; reserve: ARRAY[0..7] OF Byte END'));
001142          Field('pageRiseDirection: BOOLEAN');
001143          Field('');
001144      END;
001145      {$S SgABCres}
001146      {$ENDC}
001147
```

```
001148
001149      {$S SgABCpri}
001150      PROCEDURE {TPrintManager.}AddStripOfPages{(vhs: VHSelect)};
001151          VAR newExtentLRect: LRect;
001152              adjustment:     LONGINT;
001153      BEGIN
001154          {$IFC fTrace}BP(7);{$ENDC}
001155              WITH SELF.contentLRect DO  {cd save a mote by flipping vhs just before this}
001156                  adjustment := botRight.vh[orthogonal[vhs]] - topLeft.vh[orthogonal[vhs]];
001157              WITH SELF.view.extentLRect DO
001158                  IF vhs = v THEN
001159        {$H-}         SetLRect(newExtentLRect, left, top, right + adjustment, bottom)
001160                  ELSE
001161                      SetLRect(newExtentLRect, left, top, right, bottom + adjustment); {$H+}
001162              SELF.view.Resize(newExtentLRect);
001163              SELF.RedoBreaks;
001164          {$IFC fTrace}EP;{$ENDC}
001165      END;
001166      {$S SgABCres}
001167
001168
001169      {$S SgABCpri}
001170      PROCEDURE {TPrintManager.}ChangeMargins{(margins: LRect)};
001171      BEGIN
001172          {$IFC fTrace}BP(7);{$ENDC}
001173          WITH margins DO
001174              BEGIN {$H-}
001175              left := ABS(left);
001176              top := ABS(top);
001177              right := - ABS(right);
001178              bottom := - ABS(bottom);
001179              END; {$H+}
001180          SELF.pageMargins := margins;
001181          SELF.view.panel.currentView.ReactToPrinterChange;
001182          SELF.view.panel.Invalidate;
001183          {$IFC fTrace}EP;{$ENDC}
001184      END;
001185      {$S SgABCres}
001186
001187
001188      {$S sCldInit}
001189      PROCEDURE {TPrintManager.}ClearPageBreaks{(automatic: BOOLEAN)};
001190           VAR s:          TListScanner;
001191               break:      LONGINT;
001192               vhs:        VHSelect;
001193               endOfView:  LONGINT;
001194               breakIndex:     INTEGER;
001195      BEGIN
```

```
001196          {$IFC fTrace}BP(9);{$ENDC}
001197          { Clears all page breaks of the specified kind EXCEPT for the one marking the end of the view }
001198              FOR vhs := v TO h DO
001199                  BEGIN
001200                  endOfView := SELF.view.extentLRect.botRight.vh[orthogonal[vhs]];
001201                  breakIndex := 1;
001202                  WHILE breakIndex < SELF.breaks[vhs].size DO
001203                      BEGIN
001204                      break := TpLONGINT(SELF.breaks[vhs].At(breakIndex))^;
001205                      IF (break >= 0) = automatic THEN
001206                          IF ABS(break) < endOfView THEN
001207                              BEGIN
001208                              SELF.breaks[vhs].DelAt(breakIndex);
001209                              breakIndex := breakIndex - 1;
001210                              END;
001211                      breakIndex := breakIndex + 1;
001212                      END;
001213                  END;
001214          {$IFC fTrace}EP;{$ENDC}
001215      END;
001216      {$S SgABCres}
001217
001218
001219      PROCEDURE {TPrintManager.}DrawBreaks{(manualOnly: BOOLEAN)};
001220          VAR wLPt1:          LPoint;
001221              wLPt2:          LPoint;
001222              vhs:            VHSelect;
001223              dir:            VHSelect;
001224              viewEnd:        LONGINT;
001225              visEnd:         LONGINT;
001226              widthAdjust:    INTEGER;
001227              showing:        BOOLEAN;
001228              limit:          LONGINT;
001229              pageBreak:      LONGINT;
001230              breakIndex:     INTEGER;
001231              finished:       BOOLEAN;
001232      BEGIN
001233          {$IFC fTrace}BP(9);{$ENDC}
001234              thePad.DistToLDist(autoBreakPen.pnSize, wLPt1);
001235              thePad.DistToLDist(manualBreakPen.pnSize, wLPt2);
001236              IF NOT amPrinting THEN
001237                  FOR vhs := v TO h DO
001238                      BEGIN
001239                    {Inhibit display of breaks to the top/left of the pane}
001240                      dir := orthogonal[vhs];
001241                      viewEnd := SELF.view.extentLRect.botRight.vh[dir];
001242                      visEnd := thePad.visLRect.botRight.vh[dir];
001243                      widthAdjust := Max(wLPt1.vh[dir], wLPt2.vh[dir]) + 1;  {add 1 in case of roundoff
```

```
001244                                                                  problems}
001245
001246                  showing := FALSE;
001247                  limit := thePad.visLRect.topLeft.vh[dir];
001248
001249                  breakIndex := 1;
001250                  finished := FALSE;
001251                  WHILE (breakIndex <= SELF.breaks[vhs].size) AND NOT finished DO
001252                      BEGIN
001253                      pageBreak := TpLONGINT(SELF.breaks[vhs].At(breakIndex))^;
001254                      IF ABS(pageBreak) >= limit THEN
001255                          BEGIN
001256                          IF NOT showing THEN    {Start displaying breaks; reset limit to where we'll stop}
001257                              limit := Min(viewEnd, visEnd + widthAdjust);
001258                          showing := ABS(pageBreak) < limit;
001259                          IF NOT showing THEN   {Stop displaying breaks}
001260                              finished := TRUE;
001261                          END;
001262                      IF showing THEN
001263                          IF NOT ( (pageBreak >= 0) AND manualOnly) THEN
001264                              SELF.DrawOneBreak(pageBreak, vhs);
001265                      breakIndex := breakIndex + 1;
001266                      END;
001267                  END;
001268          {$IFC fTrace}EP;{$ENDC}
001269      END;
001270
001271
001272      PROCEDURE {TPrintManager.}DrawOneBreak{(pageBreak: LONGINT; vhs: vhSelect)};
001273          VAR lPt1:   LPoint;
001274              lPt2:   LPoint;
001275              pt:     Point;
001276              wPt:    Point; {width of line}
001277      BEGIN
001278          {$IFC fTrace}BP(6);{$ENDC}
001279
001280          IF pageBreak >= 0 THEN
001281              thePad.SetPen(autoBreakPen)
001282          ELSE
001283              thePad.SetPen(manualBreakPen);
001284
001285          lPt1 := zeroLPt;
001286          lPt2 := SELF.view.extentLRect.botRight;
001287          lPt1.vh[orthogonal[vhs]] := ABS(pageBreak);
001288          lPt2.vh[orthogonal[vhs]] := ABS(pageBreak);
001289
001290          wPt := thePort^.pnSize;
001291          wPt.vh[vhs] := 0;
```

```
001292
001293          thePad.LPtToPt(lPt1, pt);
001294          MoveTo(pt.h - wPt.h, pt.v - wPt.v); {wPt adjustment to hang line off top/left, not bot/right}
001295
001296          thePad.LPtToPt(lPt2, pt);
001297          LineTo(pt.h - wPt.h, pt.v - wPt.v); {wPt adjustment to hang line off top/left, not bot/right}
001298          {$IFC fTrace}EP;{$ENDC}
001299      END;
001300
001301
001302      {$S SgABCpri}
001303      PROCEDURE {TPrintManager.}DrawPage;
001304          VAR    heading:         THeading;
001305                  contentRect:    Rect;
001306      BEGIN
001307          {$IFC fTrace}BP(7);{$ENDC}
001308          IF (amPrinting) AND (SELF.frameBody) THEN {client wants frame drawn on printed page}
001309              BEGIN
001310              theMarginPad.LRectToRect(SELF.contentLRect, contentRect);
001311              PenNormal;
001312              PenSize(3,2);
001313              PenMode(patOr);
001314              InsetRect(contentRect, -1, -1);
001315              FrameRect(contentRect);
001316              END;
001317
001318          SELF.pageView.Draw;  {will draw headings and possibly frame body}
001319
001320          theBodyPad.Focus;
001321          SELF.view.Draw;
001322          {$IFC fTrace}EP;{$ENDC}
001323      END;
001324      {$S SgABCres}
001325
001326
001327     {$S SgABCpri}
001328      PROCEDURE {TPrintManager.}EnterPageEditting;
001329      BEGIN
001330          {$IFC fTrace}BP(7);{$ENDC}
001331          {$IFC fTrace}EP;{$ENDC}
001332      END;
001333      {$S SgABCres}
001334
001335
001336      {$S SgABCpri}
001337      PROCEDURE {TPrintManager.}GetPageLimits{(pageNumber: LONGINT; VAR viewLRect: LRect)};
001338      { NB:
001339            The default is that page numbers go up from left-to-right, as illustrated by:
```

```
001340
001341              |--------|--------|--------|
001342              | page 1 | page 2 | page 3 |
001343              |--------|--------|--------|
001344              | page 4 | page 5 | page 6 |
001345              |--------|--------|--------|
001346
001347            This is what is obtained by leaving TPrintManager.pageRiseDirection
001348             at its default value of 'h'; to get the transpose, set pageRiseDirection 'v'
001349       }
001350
001351         VAR
001352             totalStrips:        INTEGER;  {if pageRiseDirection is h, this is the total number of column
001353                                             strips}
001354             pageRiseDirection:  VHSelect;
001355             orthoDirection:     VHSelect;
001356             strips:             Point;
001357             vhs:                VHSelect;
001358             breakArray:         TArray {OF LONGINT};
001359             strip:              INTEGER;
001360             nextLocation:       LONGINT;
001361             pageBreak:          LONGINT;
001362       BEGIN
001363         {$IFC fTrace}BP(9);{$ENDC}
001364         pageRiseDirection := SELF.pageRiseDirection;
001365         orthoDirection := orthogonal[pageRiseDirection];
001366         totalStrips := SELF.breaks[orthoDirection].size;
001367
001368         strips.vh[orthoDirection] := ((pageNumber - 1) DIV totalStrips) + 1;
001369         strips.vh[pageRiseDirection] := pageNumber - ((strips.vh[orthoDirection] - 1) * totalStrips);
001370
001371         FOR vhs := v TO h DO
001372             BEGIN
001373             breakArray := SELF.breaks[orthogonal[vhs]];
001374             strip := strips.vh[vhs];
001375
001376             IF strip = 1 THEN
001377                 nextLocation := 0
001378             ELSE
001379                 BEGIN
001380                 pageBreak := TpLONGINT(breakArray.At(strip - 1))^;
001381                 nextLocation := ABS(pageBreak);
001382                 END;
001383
001384             viewLRect.topLeft.vh[vhs] := nextLocation;
001385             pageBreak := TpLONGINT(breakArray.At(strip))^;
001386             viewLRect.botRight.vh[vhs] := ABS(pageBreak);
001387             END;
```

```
001388          {$IFC fTrace}EP;{$ENDC}
001389       END;
001390      {$S SgABCres}
001391
001392
001393      FUNCTION  {TPrintManager.}NewPageView{(object: TObject): NewPageView};
001394      BEGIN
001395          {$IFC fTrace}BP(6);{$ENDC}
001396          NewPageView := TPageView.CREATE(object, SELF.Heap, SELF);
001397          {$IFC fTrace}EP;{$ENDC}
001398      END;
001399
001400
001401      FUNCTION  {TPrintManager.}NewPaginatedView{(object: TObject): TPaginatedView};
001402      {Building Block or Client reimplements this to install own flavor of paginated view}
001403      BEGIN
001404          {$IFC fTrace}BP(6);{$ENDC}
001405          NewPaginatedView := TPaginatedView.CREATE(object, SELF.Heap, SELF.view);
001406          {$IFC fTrace}EP;{$ENDC}
001407      END;
001408
001409
001410      FUNCTION  {TPrintManager.}PageWith{(VAR lPtInView: LPoint; VAR strip: Point): LONGINT};
001411          VAR pageBreak:  LONGINT;
001412              curStrip:   INTEGER;
001413              vhs:        VHSelect;
001414              finished:   BOOLEAN;
001415      BEGIN
001416          {$IFC fTrace}BP(9);{$ENDC}
001417          LRectHaveLPt(SELF.view.extentLRect, lPtInView);
001418          FOR vhs := v TO h DO
001419              BEGIN
001420              finished := FALSE;
001421              curStrip := 1;
001422              WHILE (curStrip <= SELF.breaks[orthogonal[vhs]].size) AND NOT finished DO
001423                  BEGIN
001424                  pageBreak := TpLONGINT(SELF.breaks[orthogonal[vhs]].At(curStrip))^;
001425                  IF lPtInView.vh[vhs] <= ABS(pageBreak) THEN
001426                      BEGIN
001427                      strip.vh[orthogonal[vhs]] := curStrip;
001428                      finished := TRUE;
001429                      END
001430                  ELSE
001431                      curStrip := curStrip + 1;
001432                  END;
001433              END;
001434              PageWith := (strip.vh[SELF.pageRiseDirection] - 1) *
001435                                      SELF.breaks[orthogonal[SELF.pageRiseDirection]].size
```

```
001436                          * strip.vh[orthogonal[SELF.pageRiseDirection]];
001437          {$IFC fTrace}EP;{$ENDC}
001438      END;
001439
001440
001441   {Note:  The Pepsi and the Spring versions of the following procedure are completely different}
001442
001443  {$IFC libraryVersion <= 20}        { P E P S I }
001444      {$S SgABCpri}
001445      PROCEDURE {TPrintManager.}Print{(printPref: TPrReserve)};
001446          LABEL 1,2,3,4,5,6;  {as demanded by Print Manager}
001447          VAR scaleOne:       TScaler;
001448              pageNumber:     LONGINT;
001449              rBand:          Rect;
001450              pgsTotal:       LONGINT;
001451              printerMetrics: TPrinterMetrics;
001452              error:          INTEGER;
001453              dispatchCode:   INTEGER;  {dispatch code from LisaPrint}
001454              fSpool:         BOOLEAN;
001455              prPrfAlias:     TPrPrfAlias;
001456          BEGIN
001457          {$IFC fTrace}BP(9);{$ENDC}
001458              prPrfAlias.reserve := printPref;
001459              printerMetrics :=  SELF.printerMetrics;
001460              SetPt(ScaleOne.numerator, 1, 1);
001461              SetPt(ScaleOne.denominator, 1, 1);
001462              pgsTotal := SELF.view.MaxPageToPrint; {by default, # of rowBreaks * # of colBreaks}
001463              fSpool := TRUE;
001464  1:
001465              PrDocStart(dispatchCode, prPrfAlias.prIns, printLDSN);   {open the printer}
001466
001467              CASE PrCheckErr(dispatchCode) OF
001468                  PrGoDocStart:    BEGIN
001469                                   fSpool := FALSE;
001470                                   GOTO 1;
001471                                   END;
001472
001473                  PrGoDocEnd:      GOTO 5;
001474                  PrGoExit:        GOTO 6;
001475              END;  { case }
001476
001477             theMarginPad.Rework
001478                  (SELF.view, zeroPt, printerMetrics.res, 1,
001479                      scaleOne, printerPseudoPort);  {set up margin/body pads...}
001480
001481             pageNumber := 0;
001482
001483             REPEAT
```

```
001484                  pageNumber := pageNumber + 1;
001485
001486  2:              PrStartPage(dispatchCode);
001487                  CASE PrCheckErr(dispatchCode) OF
001488
001489                      prGoDocStart:    BEGIN
001490                                       fSpool := FALSE;
001491                                       GOTO 1;
001492                                       END;
001493
001494                      prGoStartPage:   GOTO 2;
001495
001496                      prGoEndPage:     BEGIN
001497                                       SELF.SkipPage(pageNumber); {read on to start of next page, without
001498                                                                   printing this one}
001499                                       GOTO 4;
001500                                       END;
001501
001502                      prGoDocEnd:      GOTO 5;
001503                      prGoExit:        GOTO 6;
001504
001505                      prGoCont:                {actually print the page}
001506                          BEGIN
001507                          theMarginPad.SetForPage(pageNumber, zeroPt);
001508                          WHILE PrNextBand(rBand) DO
001509                              BEGIN
001510                              theMarginPad.ClipFurtherTo(rBand);
001511                              theMarginPad.Focus;
001512                              SELF.DrawPage;
001513  3:                          PrDumpBand(dispatchCode);
001514                              CASE PrCheckErr(dispatchCode) OF
001515                                  PrGoDocStart:
001516                                                  BEGIN
001517                                                  fSpool := FALSE;
001518                                                  GOTO 1;
001519                                                  END;
001520
001521                                  PrGoStartPage:  ABCBreak('PrGoStartPage received; page #=', pageNumber);
001522                                  PrGoDumpBand :  GOTO 3;
001523                                  PrGoEndPage  :  GOTO 4;
001524                                  PrGoDocEnd   :  GOTO 5;
001525                                  PrGoExit     :  GOTO 6;
001526                                  END; { CASE }
001527                              END;    {WHILE PrNextBand}
001528                          END; {prGoCont dispatch code from prStartPage}
001529                  END; {case on Err from StartPage}
001530
001531  4:              PrEndPage(dispatchCode);
```

```
001532              CASE PrCheckErr(dispatchCode) OF
001533                 PrGoDocStart:  BEGIN
001534                                fSpool := FALSE;
001535                                GOTO 1;
001536                                END;
001537
001538                 PrGoExit:      GOTO 2;
001539              END; { case }
001540
001541         UNTIL pageNumber = pgsTotal;
001542
001543  5:        PrDocEnd(dispatchCode);
001544            CASE PrCheckErr(dispatchCode) OF
001545
001546                PrGoDocStart:  BEGIN
001547                               fSpool := FALSE;
001548                               GOTO 1;
001549                               END;
001550
001551                PrGoStartPage:  ABCBreak('PrGoStartPage received; page #=', pageNumber);
001552
001553                PrGoDocEnd   :  GOTO 5;
001554                PrGoExit     :  GOTO 6;
001555            END; { case }
001556
001557  6:
001558         {$IFC fTrace}EP;{$ENDC}
001559     END;
001560  {$S SgABCres}    { END of Pepsi-release version of TPrintmanager.Print }
001561
001562  (*****************************************************************************************************)
001563
001564  {$ELSEC}      {spring-release version of TPrintManager.Print follows}
001565     {$S SgABCpri}
001566     PROCEDURE {TPrintManager.}Print{(printPref: TPrReserve)};
001567         VAR unzoomed:       TScaler;
001568             pageNumber:     LONGINT;
001569             pgsTotal:       LONGINT;
001570             prPort:         TPrPort;
001571             prPrfAlias:     TPrPrfAlias;
001572             resPageEnd:     BOOLEAN;
001573      BEGIN
001574         {$IFC fTrace}BP(9);{$ENDC}
001575         prPrfAlias.reserve := printPref;
001576         SetPt(unzoomed.numerator, 1, 1);
001577         SetPt(unzoomed.denominator, 1, 1);
001578         pgsTotal := SELF.view.MaxPageToPrint; {by default, # of rowBreaks * # of colBreaks}
001579
```

```
001580  {$IFC LibraryVersion < 30}
001581          prPrfAlias.prPrf.prLdsn := printLDSN;
001582          IF PrDocStart(prPrfAlias.prPrf, prPort {, printLDSN} ) THEN  {open the printer}       {++}
001583              {NB ldsn param not currently in spring interface, but Eric Z says it's going back in}
001584  {$ELSEC}
001585          IF PrDocStart(prPrfAlias.prPrf, prPort , printLDSN, TRUE) THEN  {open the printer}       {++}
001586  {$ENDC}
001587              BEGIN
001588              theMarginPad.Rework
001589                   (SELF.view, zeroPt, SELF.printerMetrics.res, 1,
001590                       unzoomed, printerPseudoPort);  {set up margin/body pads...}
001591              RectRgn(altVisRgn, hugeRect);
001592              useAltVisRgn := TRUE;
001593
001594              pageNumber := 0;
001595
001596              REPEAT
001597                  pageNumber := pageNumber + 1;
001598
001599                  IF NOT PrPageStart(prPrfAlias.prPrf, prPort) THEN {+SW+}
001600                      SELF.SkipPage(pageNumber)  {read on to start of next page, without printing this one}
001601                  ELSE
001602                      BEGIN
001603                      theMarginPad.SetForPage(pageNumber, zeroPt);
001604                      theMarginPad.Focus;
001605                      SELF.DrawPage;
001606                      END;
001607                  resPageEnd := PrPageEnd(prPrfAlias.prPrf, prPort);
001608              UNTIL
001609                  resPageEnd OR (pageNumber >= pgsTotal);
001610
001611              PrDocEnd(prPrfAlias.prPrf, prPort);
001612              {??? Do we need to stuff the prRec back into the doc?? Must ask Bayles}
001613
001614              useAltVisRgn := FALSE;
001615              END;
001616          {$IFC fTrace}EP;{$ENDC}
001617      END;
001618  {$S SgABCres}
001619  {$ENDC}  {End of Spring-Release version of TPrintManager.Print}
001620
001621
001622      {$S sCldInit}
001623      PROCEDURE {TPrintManager.}ReactToPrinterChange;      {several changes}
001624          VAR newExtent:      LRect;
001625              minViewLRect:   LRect;
001626              s:              TListScanner;
001627              pageBreak:      LONGINT;
```

```
001628              vhs:            VHSelect;
001629              curLPt:         LPoint;
001630              pageIncrement:  LONGINT;
001631              metrics:        TPrinterMetrics;
001632
001633          PROCEDURE ScaleToViewedSpace(printRect: Rect; VAR viewedLRect: LRect);
001634              BEGIN
001635                  SetLRect(viewedLRect,
001636                    LIntOvrInt(ORD4(printRect.left)   * SELF.view.res.h, metrics.res.h),
001637                    LIntOvrInt(ORD4(printRect.top)    * SELF.view.res.v, metrics.res.v),
001638                    LIntOvrInt(ORD4(printRect.right)  * SELF.view.res.h, metrics.res.h),
001639                    LIntOvrInt(ORD4(printRect.bottom) * SELF.view.res.v, metrics.res.v));
001640              END;
001641
001642       BEGIN
001643          {$IFC fTrace}BP(9);{$ENDC}
001644           { SELF.InvalidatePageBreaks, or some such ???}
001645           SELF.view.panel.window.GetPrinterMetrics;  {except maybe for view in first Panel created, this will
001646                                                       be an unnecessary (but inexpensive) step}
001647           metrics := SELF.view.panel.window.printerMetrics;
001648           SELF.printerMetrics := metrics;
001649
001650           WITH SELF, printerMetrics DO
001651               BEGIN
001652    {$H-}          ScaleToViewedSpace(paperRect,  paperLRect);
001653               ScaleToViewedSpace(printRect, printableLRect);
001654               END;
001655          LRectPlusLRect(SELF.paperLRect, SELF.pageMargins, SELF.contentLRect); {$H+}
001656
001657          SELF.pageView.Resize(SELF.paperLRect);
001658          SELF.view.SetMinViewSize(newExtent);  {++}
001659          SELF.view.Resize(newExtent);           {set view back to its min size}
001660          SELF.RedoBreaks;                    {may resize the view upwards again by a bit}
001661         {SELF.InvalidatePageBreaks again -- to force update where new breaks are to be shown}
001662       {$IFC fTrace}EP;{$ENDC}
001663      END;
001664      {$S SgABCres}
001665
001666
001667      {$S sCldInit}
001668      PROCEDURE {TPrintManager.}RedoBreaks;
001669           VAR vhs:                  VHSelect;
001670               maxViewPixelsPerPage: INTEGER;
001671               curLocation:          LONGINT;
001672               onePixelTooMuch:      LONGINT;
001673               endOfView:            LONGINT;
001674               s:                    TListScanner;
001675               nextPageBreak:        LONGINT;
```

```
001676              breakIndex:           INTEGER;
001677              penultimatePageBreak: LONGINT;
001678              newViewExtent:        LRect;
001679      BEGIN
001680          {$IFC fTrace}BP(9);{$ENDC}
001681          newViewExtent := SELF.view.extentLRect;
001682          SELF.ClearPageBreaks(TRUE);  {clear out old automatic breaks}
001683          FOR vhs := v TO h DO
001684              BEGIN
001685              WITH SELF.contentLRect DO
001686                  IF vhs = v THEN
001687                      maxViewPixelsPerPage := right - left
001688                  ELSE
001689                      maxViewPixelsPerPage := bottom - top;
001690
001691              endOfView := SELF.view.extentLRect.botRight.vh[orthogonal[vhs]];
001692
001693              breakIndex := 1;
001694              curLocation := 0;
001695              WHILE curLocation < endOfView DO
001696                  BEGIN
001697                  nextPageBreak := TpLONGINT(SELF.breaks[vhs].At(breakIndex))^;
001698                  onePixelTooMuch := Min(curLocation + MaxViewPixelsPerPage, endOfView);
001699                  IF ABS(nextPageBreak) <= onePixelTooMuch THEN
001700                      curLocation := ABS(nextPageBreak)
001701                  ELSE {no manual page break; impose an automatic one -- propose onePixelTooMuch}
001702                      BEGIN
001703                      curLocation := SELF.view.ForceBreakAt(vhs, curLocation, onePixelTooMuch);
001704                      SELF.breaks[vhs].InsAt(breakIndex, @curLocation);
001705                      END;
001706                  breakIndex := breakIndex + 1;
001707                  END;
001708
001709              IF SELF.view.fitPagesPerfectly THEN  {make minor adjustment upward}
001710                  BEGIN
001711                  IF (SELF.breaks[vhs].size > 1) THEN
001712                      penultimatePageBreak := TpLONGINT(SELF.breaks[vhs].At(SELF.breaks[vhs].size - 1))^
001713                  ELSE
001714                      penultimatePageBreak := 0;
001715                  newViewExtent.botRight.vh[orthogonal[vhs]] := ABS(penultimatePageBreak) +
001716                                                                maxViewPixelsPerPage;
001717                  END;
001718
001719              END; {for vhs := v to h}
001720          SELF.view.Resize(newViewExtent);
001721          {$IFC fTrace}EP;{$ENDC}
001722      END;
001723      {$S SgABCres}
```

```
001724
001725
001726      {$S SgABCpri}
001727      PROCEDURE {TPrintManager.}SetBreak{(vhs: VHSelect; where: LONGINT; isAutomatic: BOOLEAN)};
001728           VAR s:             TListScanner;
001729               break:         LONGINT;     {comment gone}
001730               prevBreakLoc:  LONGINT;
001731               breakIndex:    INTEGER;
001732               finished:      BOOLEAN;
001733      BEGIN
001734          {$IFC fTrace}BP(9);{$ENDC}
001735          prevBreakLoc := 0;
001736          breakIndex := 1;
001737          finished := FALSE;
001738          WHILE (breakIndex <= SELF.breaks[vhs].size) AND NOT finished DO
001739              BEGIN
001740              break := TpLONGINT(SELF.breaks[vhs].At(breakIndex))^;
001741              IF ABS(break) > where THEN
001742                {found where to insert!}
001743                  BEGIN
001744                  where := SELF.view.ForceBreakAt(vhs, prevBreakLoc, where);
001745                  break := where;
001746                  IF NOT isAutomatic THEN
001747                      break := - break;
001748                  SELF.breaks[vhs].InsAt(breakIndex, @break);
001749                  finished := TRUE;
001750                  END
001751              ELSE
001752              IF ABS(break) = where THEN
001753                 {replace an existing page break}
001754                  BEGIN
001755                  break := where;
001756                  IF NOT isAutomatic THEN
001757                      break := - break;
001758                  SELF.breaks[vhs].PutAt(breakIndex, @break);
001759                  finished := TRUE;
001760                  END
001761              ELSE
001762                  prevBreakLoc := ABS(break);
001763
001764              breakIndex := breakIndex + 1;
001765              END;
001766          {$IFC fTrace}EP;{$ENDC}
001767      END;
001768      {$S SgABCres}
001769
001770
001771      {$S SgABCcld}
```

```
001772      PROCEDURE  {TPrintManager.}SetDfltHeadings;  {client redefines}
001773      BEGIN
001774          {$IFC fTrace}BP(7);{$ENDC}
001775          {$IFC fTrace}EP;{$ENDC}
001776      END;
001777      {$S SgABCres}
001778
001779
001780      {$S SgABCpri}
001781       PROCEDURE {TPrintManager.}SkipPage{(pageNumber: LONGINT)};  {client may want to redefine}
001782       BEGIN
001783          {$IFC fTrace}BP(9);{$ENDC}
001784          {$IFC fTrace}EP;{$ENDC}
001785       END;
001786      {$S SgABCres}
001787
001788
001789  {$S SgABCini}
001790  END;
001791  {$S SgABCres}
001792
001793
001794
001795  METHODS OF THeading;
001796
001797
001798  {$S SgABCini}
001799      FUNCTION  {THeading.}CREATE{(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
001800            itsExtentLRect: LRect; itsPageAlignment: TPageAlignment; itsOffsetFromAlignment: LPoint): THeading};
001801      BEGIN
001802          {$IFC fTrace}BP(6);{$ENDC}
001803          IF object = NIL THEN
001804              object := NewObject(heap, THISCLASS);
001805          SELF := THeading(TImage.CREATE(object, heap, itsExtentLRect, itsPrintManager.pageView));
001806
001807          WITH SELF DO
001808              BEGIN
001809              printManager := itsPrintManager;
001810              pageAlignment := itsPageAlignment;
001811              offsetFromAlignment := itsOffsetFromAlignment;
001812              oddOnly := FALSE;
001813              evenOnly := FALSE;
001814              minPage := 2;
001815              maxPage := MAXLINT;
001816              END;
001817          {$IFC fTrace}EP;{$ENDC}
001818      END;
001819
```

```
001820
001821      {$IFC fDebugMethods}
001822      {$S SgABCdbg}
001823      PROCEDURE {THeading.}Fields{(PROCEDURE Field(nameAndType: S255))};
001824      BEGIN
001825          TImage.Fields(Field);
001826          Field('printManager: TPrintManager');
001827          Field('pageAlignment: Byte'); {enumerated type}
001828          Field('offsetFromAlignment: LPoint');
001829          Field('oddOnly: BOOLEAN');
001830          Field('evenOnly: BOOLEAN');
001831          Field('minPage: LONGINT');
001832          Field('maxPage: LONGINT');
001833          Field('');
001834      END;
001835      {$S SgABCcld}
001836      {$ENDC}
001837
001838
001839      PROCEDURE {THeading.}AdjustForPage{(pageNumber: LONGINT; editing: BOOLEAN)};
001840      {will be overridden in Subclass if meaningful}
001841      BEGIN
001842          {$IFC fTrace}BP(9);{$ENDC}
001843          {$IFC fTrace}EP;{$ENDC}
001844      END;
001845
001846
001847      PROCEDURE {THeading.}ChangePageAlignment{(newPageAlignment: TPageAlignment)};
001848          VAR newOffset:      LPoint;
001849          FUNCTION Mid(anLRect: LRect; vhs: VHSelect): LONGINT;
001850              BEGIN
001851              Mid := (anLRect.topLeft.vh[vhs] + anLRect.botRight.vh[vhs]) DIV 2;
001852              END;
001853      BEGIN
001854          {$IFC fTrace}BP(9);{$ENDC}
001855          IF SELF.pageAlignment <> newPageAlignment THEN
001856              BEGIN
001857              CASE newPageAlignment OF
001858                  aTopLeft, aBottomLeft:
001859                      newOffset.h := SELF.extentLRect.left -  SELF.view.extentLRect.left;
001860                  aTopCenter, aBottomCenter:
001861                      newOffset.h := Mid(SELF.extentLRect, h)  -  Mid(SELF.view.extentLRect, h);
001862                  aTopRight, aBottomRight:
001863                      newOffset.h := SELF.extentLRect.right - SELF.view.extentLRect.right;
001864                  END;
001865              CASE newPageAlignment OF
001866                  aTopLeft, aTopCenter, aTopRight:
001867                      newOffset.v := SELF.extentLRect.top - SELF.view.extentLRect.top;
```

```
001868                  aBottomLeft, aBottomCenter, aBottomRight:
001869                      newOffset.v := SELF.extentLRect.bottom - SELF.view.extentLRect.bottom;
001870                  END;
001871              SELF.offsetFromAlignment := newOffset;
001872              SELF.pageAlignment := newPageAlignment;
001873              END;
001874          {$IFC fTrace}EP;{$ENDC}
001875      END;
001876
001877
001878      PROCEDURE {THeading.}Draw;  {will be overridden in Subclass if meaningful}
001879      BEGIN
001880          {$IFC fTrace}BP(9);{$ENDC}
001881          IF SELF.shouldFrame THEN
001882              FrameLRect(SELF.extentLRect);
001883          {$IFC fTrace}EP;{$ENDC}
001884      END;
001885
001886
001887      PROCEDURE {THeading.}LocateOnPage{(editing: BOOLEAN)};
001888      {called after client has adjusted the extentLRect and (possibly) the offsetFromAlignment}
001889          VAR currentH, currentV, targetH, targetV:   LONGINT;
001890              offset: LPoint;
001891              pmgr:   TPrintManager;              (* CIRCUMVENT COMPILER BUG *)
001892          {NB: Someday someone could use vhs and other tricks to tighten this up}
001893      BEGIN
001894          {$IFC fTrace}BP(9);{$ENDC}
001895          WITH SELF DO
001896            BEGIN         (* CIRCUMVENT COMPILER BUG *)
001897            pmgr := SELF.printManager;          (* CIRCUMVENT COMPILER BUG *)
001898            WITH pmgr, paperLRect DO
001899              BEGIN
001900              CASE {SELF.}pageAlignment OF
001901                  aTopLeft,
001902                    aBottomLeft:      BEGIN
001903                                      currentH := {SELF.}extentLRect.left;
001904                                      targetH := {paperLRect.}left;
001905                                      END;
001906                  aTopCenter,
001907                    aBottomCenter:    BEGIN
001908                                      currentH := (extentLRect.right + extentLRect.left) DIV 2;
001909                                      targetH := {paperLRect.}(right + left) DIV 2;
001910                                      END;
001911                  aTopRight,
001912                    aBottomRight:     BEGIN
001913                                      currentH := extentLRect.right;
001914                                      targetH :=  right;
001915                                      END;
```

```
001916                END;
001917
001918            CASE {SELF.}pageAlignment OF
001919                aTopLeft,
001920                  aTopCenter,
001921                  aTopRight:        BEGIN
001922                                    currentV := extentLRect.top;
001923                                    targetV := top;
001924                                    END;
001925                aBottomLeft,
001926                  aBottomCenter,
001927                  aBottomRight:     BEGIN
001928                                    currentV := extentLRect.bottom;
001929                                    targetV := bottom;
001930                                    END;
001931                END;
001932            END;
001933          END;         (* CIRCUMVENT COMPILER BUG *)
001934
001935        WITH SELF.offsetFromAlignment DO
001936            {$H-}SetLPt(offset, targetH - currentH + h, targetV - currentV + v); {$H+}
001937        SELF.OffsetBy(offset);
001938        {$IFC fTrace}EP;{$ENDC}
001939    END;
001940
001941
001942    FUNCTION  {THeading.}ShouldDraw{(pageNumber: LONGINT): BOOLEAN};
001943        VAR judgment: BOOLEAN;
001944    BEGIN
001945        {$IFC fTrace}BP(9);{$ENDC}
001946        WITH SELF DO
001947            IF (oddOnly AND NOT ODD(pageNumber)) OR
001948               (evenOnly AND ODD(pageNumber))    OR
001949               (pageNumber < minPage)            OR
001950               (pageNumber > maxPage)    THEN
001951                    judgment := FALSE
001952            ELSE
001953               judgment := TRUE;
001954        ShouldDraw := judgment;
001955        {$IFC fTrace}EP;{$ENDC}
001956    END;
001957
001958
001959    FUNCTION  {THeading.}ShouldFrame{: BOOLEAN};
001960    BEGIN
001961        {$IFC fTrace}BP(9);{$ENDC}
001962        ShouldFrame := NOT amPrinting;  {default}
001963        {$IFC fTrace}EP;{$ENDC}
```

```
001964      END;
001965
001966
001967  {$S SgABCini}
001968  END;
001969  {$S SgABCres}
001970
001971
001972
001973  METHODS OF TSelection;
001974
001975      {$S sStartup}
001976      FUNCTION  {TSelection.}CREATE{(object: TObject; heap: THeap; itsView: TView; itsKind: INTEGER;
001977                                     itsAnchorLPt: LPoint): TSelection};
001978      BEGIN
001979          {$IFC fTrace}BP(7);{$ENDC}
001980          IF object = NIL THEN
001981              object := NewObject(heap, THISCLASS);
001982          SELF := TSelection(object);
001983
001984          WITH SELF DO
001985              BEGIN
001986              currLPt := itsAnchorLPt;
001987              anchorLPt := itsAnchorLPt;
001988              boundLRect := hugeLRect;
001989              kind := itsKind;
001990              view := itsView;
001991              panel := view.panel;
001992              IF panel <> NIL THEN
001993                  window := panel.window;
001994              coSelection := NIL;
001995              canCrossPanels := FALSE;
001996              END;
001997          {$IFC fTrace}EP;{$ENDC}
001998      END;
001999
002000
002001      {$S sRes}
002002      FUNCTION  {TSelection.}Clone{(heap: Theap): TObject};
002003      VAR selection:      TSelection;
002004          coSelection:    Tselection;
002005      BEGIN
002006          {$IFC fTrace}BP(7);{$ENDC}
002007          selection := TSelection(SUPERSELF.Clone(heap));
002008          IF SELF.coSelection <> NIL THEN
002009              BEGIN
002010              coSelection := TSelection(SELF.coSelection.Clone(heap));
002011              selection.coSelection := coSelection;
```

```
002012              END;
002013          Clone := selection;
002014          {$IFC fTrace}EP;{$ENDC}
002015      END;
002016
002017
002018      {$S sRes}
002019      PROCEDURE {TSelection.}Free;
002020      BEGIN
002021          {$IFC fTrace}BP(7);{$ENDC}
002022          Free(SELF.coSelection);
002023          TObject.Free;
002024          {$IFC fTrace}EP;{$ENDC}
002025      END;
002026
002027
002028      {$S sRes}
002029      FUNCTION  {TSelection.}FreedAndReplacedBy{(selection: TSelection): TSelection};
002030      BEGIN
002031          {$IFC fTrace}BP(7);{$ENDC}
002032          SELF.Become(selection);
002033          FreedAndReplacedBy := SELF;
002034          {$IFC fTrace}EP;{$ENDC}
002035      END;
002036
002037
002038      {$IFC fDebugMethods}
002039      {$S SgABCdbg}
002040      PROCEDURE {TSelection.}Fields{(PROCEDURE Field(nameAndType: S255))};
002041      BEGIN
002042          Field('window: TWindow');
002043          Field('panel: TPanel');
002044          Field('view: TView');
002045          Field('kind: INTEGER');
002046          Field('anchorLPt: LPoint');
002047          Field('currLPt: LPoint');
002048          Field('boundLRect: LRect'); {+++LSR+++}
002049          Field('coSelection: TSelection');
002050          Field('canCrossPanels: BOOLEAN');
002051          Field('');
002052      END;
002053      {$S SgABCres}
002054      {$ENDC}
002055
002056
002057      {$S sRes}
002058      FUNCTION  {TSelection.}CanDoCommand{(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN};
002059      BEGIN
```

```
002060          {$IFC fTrace}BP(6);{$ENDC}
002061          IF SELF.coSelection <> NIL THEN
002062              CanDoCommand := SELF.coSelection.CanDoCommand(cmdNumber, checkIt)
002063          ELSE
002064              CanDoCommand := SELF.window.CanDoCommand(cmdNumber, checkIt);
002065          {$IFC fTrace}EP;{$ENDC}
002066      END;
002067
002068
002069      {$S sAlert}
002070      PROCEDURE {TSelection.}CantDoCmd{(cmdNumber: TCmdNumber)};
002071          VAR cmdStr: S255;
002072      BEGIN
002073          {$IFC fTrace}BP(7);{$ENDC}
002074          IF menuBar.GetCmdName(cmdNumber, @cmdStr) THEN
002075              BEGIN
002076              process.ArgAlert(1, cmdStr);
002077              process.Stop(phUnkCmd);
002078              END
002079          ELSE
002080              SELF.CantDoIt;
002081          {$IFC fTrace}EP;{$ENDC}
002082      END;
002083
002084
002085      {$S sAlert}
002086      PROCEDURE {TSelection.}CantDoIt;
002087          VAR ph: INTEGER;
002088      BEGIN
002089          {$IFC fTrace}BP(7);{$ENDC}
002090          IF SELF.kind = nothingKind THEN
002091              ph := phNoSel
002092          ELSE
002093              ph := phSelCant;
002094          process.Stop(ph);
002095          {$IFC fTrace}EP;{$ENDC}
002096      END;
002097
002098
002099      {$S sRes}
002100      PROCEDURE {TSelection.}Deselect;
002101      VAR selection:  TSelection;
002102      BEGIN
002103          {$IFC fTrace}BP(7);{$ENDC}
002104          SELF.panel.Highlight(SELF, hOnToOff);
002105          selection := SELF.FreedAndReplacedBy(SELF.view.NoSelection);
002106          {$IFC fTrace}EP;{$ENDC}
002107      END;
```

```
002108
002109
002110      {$S SgABCcld}
002111      PROCEDURE {TSelection.}DrawGhost;
002112      BEGIN
002113          {$IFC fTrace}BP(7);{$ENDC}
002114          {$IFC fTrace}EP;{$ENDC}
002115      END;
002116
002117
002118      {$S sRes}
002119      PROCEDURE {TSelection.}DoKey{(ascii: CHAR; keycap: Byte; shiftKey, appleKey, optionKey: BOOLEAN)};
002120          VAR cmdNumber:    TCmdNumber;
002121      BEGIN
002122          {$IFC fTrace}BP(7);{$ENDC}
002123          IF appleKey THEN
002124              BEGIN
002125              SELF.window.SetupMenus;
002126              cmdNumber := menuBar.CmdKey(ascii);
002127              SELF.window.DoCommand(cmdNumber);
002128              END
002129          ELSE
002130          IF currentDocument = clipboard THEN
002131              process.Stop(phEditClip)
002132          ELSE
002133          IF SELF.kind = nothingKind THEN
002134              process.Stop(phNoSel)
002135          ELSE
002136              BEGIN
002137              CASE ORD(ascii) OF
002138                  ascArwDown:
002139                      SELF.KeyEnter(0, 1);
002140                  ascArwLeft:
002141                      SELF.KeyEnter(-1, 0);
002142                  ascArwRight:
002143                      SELF.KeyEnter(1, 0);
002144                  ascArwUp:
002145                      SELF.KeyEnter(0, -1);
002146                  ascClear:
002147                      SELF.KeyClear;
002148                  ascEnter:
002149                      SELF.KeyEnter(0, 0);
002150                  OTHERWISE
002151                      CASE ORD(ascii) OF
002152                          ascBackspace:
002153                              IF shiftKey THEN
002154                                  SELF.KeyForward(appleKey)
002155                              ELSE
```

```
002156                              SELF.KeyBack(appleKey);
002157                          ascReturn:
002158                              SELF.KeyReturn;
002159                          ascTab:
002160                              SELF.KeyTab(shiftKey);
002161                          OTHERWISE
002162                              SELF.KeyChar(ascii);
002163                          END;
002164                  END;
002165              IF ORD(ascii) <> ascClear THEN
002166                  process.RememberCommand(uKeyDown); {clear is special}
002167              END;
002168          {$IFC fTrace}EP;{$ENDC}
002169      END;
002170
002171
002172      {$S sRes}
002173      PROCEDURE {TSelection.}GetHysteresis{(VAR hysterPt: Point)};
002174      BEGIN
002175          {$IFC fTrace}BP(3);{$ENDC}
002176          SetPt(hysterPt, stdHHysteresis, stdVHysteresis);
002177          {$IFC fTrace}EP;{$ENDC}
002178      END;
002179
002180
002181      {$S SgABCpri}
002182      PROCEDURE {TSelection.}HaveView{(view: TView)};
002183      BEGIN
002184          {$IFC fTrace}BP(7);{$ENDC}
002185          SELF.view := view;
002186          IF SELF.coSelection <> NIL THEN
002187              SELF.coSelection.HaveView(view);
002188          {$IFC fTrace}EP;{$ENDC}
002189      END;
002190
002191
002192      {$S sStartup}
002193      PROCEDURE {TSelection.}Highlight{(highTransit: THighTransit)};
002194      BEGIN
002195          {$IFC fTrace}BP(7);{$ENDC}
002196          IF SELF.coSelection <> NIL THEN
002197              SELF.coSelection.Highlight(highTransit);
002198          {$IFC fTrace}EP;{$ENDC}
002199      END;
002200
002201
002202      {$S sRes}
002203      PROCEDURE {TSelection.}IdleBegin{(centiSeconds: LONGINT)};
```

```
002204      BEGIN
002205          {$IFC fTrace}BP(7);{$ENDC}
002206          IF SELF.coSelection <> NIL THEN
002207              SELF.coSelection.IdleBegin(centiSeconds)
002208          ELSE
002209              SELF.window.IdleBegin(centiSeconds);
002210          {$IFC fTrace}EP;{$ENDC}
002211      END;
002212
002213
002214      {$S sRes}
002215      PROCEDURE {TSelection.}IdleContinue{(centiSeconds: LONGINT)};
002216      BEGIN
002217          {$IFC fTrace}BP(5);{$ENDC}
002218          IF SELF.coSelection <> NIL THEN
002219              SELF.coSelection.IdleContinue(centiSeconds)
002220          ELSE
002221              SELF.window.IdleContinue(centiSeconds);
002222          {$IFC fTrace}EP;{$ENDC}
002223      END;
002224
002225
002226      {$S sRes}
002227      PROCEDURE {TSelection.}IdleEnd{(centiSeconds: LONGINT)};
002228      BEGIN
002229          {$IFC fTrace}BP(7);{$ENDC}
002230          IF SELF.coSelection <> NIL THEN
002231              SELF.coSelection.IdleEnd(centiSeconds)
002232          ELSE
002233              SELF.window.IdleEnd(centiSeconds);
002234          {$IFC fTrace}EP;{$ENDC}
002235      END;
002236
002237
002238      {$S sRes}
002239      PROCEDURE {TSelection.}KeyBack{(fWord: BOOLEAN)};
002240      BEGIN
002241          {$IFC fTrace}BP(7);{$ENDC}
002242          IF SELF.coSelection <> NIL THEN
002243              SELF.coSelection.KeyBack(fWord)
002244          ELSE
002245              SELF.CantDoCmd(uBackspace);
002246          {$IFC fTrace}EP;{$ENDC}
002247      END;
002248
002249
002250      {$S sRes}
002251      PROCEDURE {TSelection.}KeyChar{(ch: CHAR)};
```

```
002252      BEGIN
002253          {$IFC fTrace}BP(7);{$ENDC}
002254          IF SELF.coSelection <> NIL THEN
002255              SELF.coSelection.KeyChar(ch)
002256          ELSE
002257              SELF.CantDoCmd(uTyping);
002258          {$IFC fTrace}EP;{$ENDC}
002259      END;
002260
002261
002262      {$S sRes}
002263      PROCEDURE {TSelection.}KeyClear;
002264          VAR dummy:  BOOLEAN;
002265      BEGIN
002266          {$IFC fTrace}BP(7);{$ENDC}
002267          IF SELF.coSelection <> NIL THEN
002268              SELF.coSelection.KeyClear
002269          ELSE IF (menubar.GetCmdName(uClear, NIL))   {there is a CLEAR menu item} AND
002270                  (SELF.CanDoCommand(uClear, dummy))  {the selection says it can do it} THEN
002271              BEGIN                                   {make believe the user chose the menu item}
002272              menuBar.HighlightMenu(uClear);
002273              SELF.window.DoCommand(uClear);
002274              END
002275          ELSE
002276              BEGIN
002277              SELF.CantDoCmd(uClear);
002278              process.RememberCommand(uClear);
002279              END;
002280          {$IFC fTrace}EP;{$ENDC}
002281      END;
002282
002283
002284      {$S sRes}
002285      PROCEDURE {TSelection.}KeyEnter{(dh, dv: INTEGER)};
002286      BEGIN
002287          {$IFC fTrace}BP(7);{$ENDC}
002288          IF SELF.coSelection <> NIL THEN
002289              SELF.coSelection.KeyEnter(dh, dv)
002290          ELSE
002291              SELF.CantDoCmd(uEnter);
002292          {$IFC fTrace}EP;{$ENDC}
002293      END;
002294
002295
002296      {$S sRes}
002297      PROCEDURE {TSelection.}KeyForward{(fWord: BOOLEAN)};
002298      BEGIN
002299          {$IFC fTrace}BP(7);{$ENDC}
```

```
002300          IF SELF.coSelection <> NIL THEN
002301              SELF.coSelection.KeyForward(fWord)
002302          ELSE
002303              SELF.CantDoCmd(uForwardSpace);
002304          {$IFC fTrace}EP;{$ENDC}
002305      END;
002306
002307
002308      {$S sRes}
002309      PROCEDURE {TSelection.}KeyPause;
002310      BEGIN
002311          {$IFC fTrace}BP(7);{$ENDC}
002312          IF SELF.coSelection <> NIL THEN
002313              SELF.coSelection.KeyPause;
002314          {$IFC fTrace}EP;{$ENDC}
002315      END;
002316
002317
002318      {$S sRes}
002319      PROCEDURE {TSelection.}KeyReturn;
002320      BEGIN
002321          {$IFC fTrace}BP(7);{$ENDC}
002322          IF SELF.coSelection <> NIL THEN
002323              SELF.coSelection.KeyReturn
002324          ELSE
002325              SELF.CantDoCmd(uReturn);
002326          {$IFC fTrace}EP;{$ENDC}
002327      END;
002328
002329
002330      {$S sRes}
002331      PROCEDURE {TSelection.}KeyTab{(fBackward: BOOLEAN)};
002332      BEGIN
002333          {$IFC fTrace}BP(7);{$ENDC}
002334          IF SELF.coSelection <> NIL THEN
002335              SELF.coSelection.KeyTab(fBackward)
002336          ELSE
002337              SELF.CantDoCmd(uTab);
002338          {$IFC fTrace}EP;{$ENDC}
002339      END;
002340
002341
002342      {$S SgABCres}
002343      PROCEDURE {TSelection.}MarkChanged;
002344          VAR delta:      INTEGER;
002345      BEGIN
002346          {$IFC fTrace}BP(7);{$ENDC}
002347          IF SELF.panel.window = currentWindow THEN
```

```
002348              BEGIN
002349              IF currentWindow.lastCmd = NIL THEN
002350                  delta := 1
002351              ELSE
002352              IF currentWindow.lastCmd.doing THEN
002353                  delta := 1
002354              ELSE
002355                  delta := -1;
002356
002357              currentWindow.changes := currentWindow.changes + delta;
002358              IF boundDocument = currentDocument THEN
002359                  WITH boundDocument DO
002360                      dataSegment.changes := dataSegment.changes + delta;
002361              END;
002362          {$IFC fTrace}EP;{$ENDC}
002363      END;
002364
002365
002366      {$S sRes}
002367      PROCEDURE {TSelection.}MousePress{(mouseLPt: LPoint)};
002368      BEGIN
002369          {$IFC fTrace}BP(7);{$ENDC}
002370          IF SELF.coSelection <> NIL THEN
002371              SELF.coSelection.MousePress(mouseLPt);
002372          {$IFC fTrace}EP;{$ENDC}
002373      END;
002374
002375
002376      {$S sRes}
002377      PROCEDURE {TSelection.}MouseMove{(mouseLPt: LPoint)};
002378      BEGIN
002379          {$IFC fTrace}BP(7);{$ENDC}
002380          IF SELF.coSelection <> NIL THEN
002381              SELF.coSelection.MouseMove(mouseLPt);
002382          {$IFC fTrace}EP;{$ENDC}
002383      END;
002384
002385
002386      {$S sRes}
002387      PROCEDURE {TSelection.}MouseRelease;
002388      BEGIN
002389          {$IFC fTrace}BP(7);{$ENDC}
002390          IF SELF.coSelection <> NIL THEN
002391              SELF.coSelection.MouseRelease;
002392          {$IFC fTrace}EP;{$ENDC}
002393      END;
002394
002395
```

```
002396      {$S SgABCcld}
002397      PROCEDURE {TSelection.}MoveBackToAnchor;        {dest panel of cross-panel drag refused DoReceive}
002398      BEGIN
002399          {$IFC fTrace}BP(7);{$ENDC}
002400          {$IFC fTrace}EP;{$ENDC}
002401      END;
002402
002403
002404      {$S sRes}
002405      FUNCTION  {TSelection.}NewCommand{(cmdNumber: TCmdNumber): TCommand};
002406      BEGIN
002407          {$IFC fTrace}BP(7);{$ENDC}
002408          IF SELF.coSelection <> NIL THEN
002409              NewCommand := SELF.coSelection.NewCommand(cmdNumber)
002410          ELSE
002411              NewCommand := SELF.window.NewCommand(cmdNumber);
002412          {$IFC fTrace}EP;{$ENDC}
002413      END;
002414
002415
002416      {$S sRes}
002417      PROCEDURE {TSelection.}PerformCommand{(command: TCommand; cmdPhase: TCmdPhase)};{+sw+}
002418      BEGIN
002419          {$IFC fTrace}BP(7);{$ENDC}
002420          command.doing := (cmdPhase <> undoPhase);
002421          command.Perform(cmdPhase);
002422          {$IFC fTrace}EP;{$ENDC}
002423      END;
002424
002425
002426      {$S sRes}
002427      PROCEDURE {TSelection.}Restore;     {SELF should be undoSelection}
002428          VAR selection:  TSelection;
002429      BEGIN
002430          {$IFC fTrace}BP(7);{$ENDC}
002431          selection := SELF.panel.selection.FreedAndReplacedBy(
002432                                              TSelection(SELF.panel.undoSelection.Clone(SELF.Heap))); {$}
002433          selection := SELF.panel.undoSelection.FreedAndReplaceBy(SELF.view.NoSelection);
002434          {$IFC fTrace}EP;{$ENDC}
002435      END;
002436
002437
002438      {$S sRes}
002439      PROCEDURE {TSelection.}Reveal(asMuchAsPossible: BOOLEAN);
002440          TYPE TXLRect = PACKED ARRAY [1..SIZEOF(LRect)] OF CHAR;
002441          VAR lr:         LRect;
002442              hMin:       INTEGER;
002443              vMin:       INTEGER;
```

```
002444      BEGIN
002445          {$IFC fTrace}BP(7);{$ENDC}
002446          IF SELF.coSelection <> NIL THEN
002447              SELF.coSelection.Reveal(asMuchAsPossible)
002448          ELSE
002449              BEGIN
002450              lr := SELF.boundLRect;
002451              IF TXLRect(lr) <> TXLRect(hugeLRect) THEN
002452                  BEGIN
002453                  IF NOT asMuchAsPossible THEN
002454                      BEGIN
002455                      hMin := 30;
002456                      vMin := 20;
002457                      END
002458                  ELSE
002459                      WITH lr DO
002460                          BEGIN
002461                          hMin := Min(MAXINT, right - left + 6);
002462                          vMin := Min(MAXINT, bottom - top + 4);
002463                          END;
002464                  SELF.panel.RevealLRect(lr, hMin, vMin);
002465                  END;
002466              END;
002467          {$IFC fTrace}EP;{$ENDC}
002468      END;
002469
002470
002471      {$S sRes}
002472      PROCEDURE {TSelection.}Save;
002473          VAR selection:  TSelection;
002474      BEGIN
002475          {$IFC fTrace}BP(7);{$ENDC}
002476          selection := SELF.panel.undoSelection.FreedAndReplacedBy(TSelection(SELF.Clone(SELF.Heap)));
002477          {$IFC fTrace}EP;{$ENDC}
002478      END;
002479
002480
002481      {$S sRes}
002482      PROCEDURE {TSelection.}SelectParagraphs;
002483      BEGIN
002484          {$IFC fTrace}BP(7);{$ENDC}
002485          IF SELF.coSelection <> NIL THEN
002486              SELF.coSelection.SelectParagraphs;
002487          {$IFC fTrace}EP;{$ENDC}
002488      END;
002489
002490
002491  {$S SgABCini}
```

```
002492  BEGIN
002493      cSelection := THISCLASS;
002494  END;
002495  {$S SgABCres}
002496
002497
002498
002499
002500
002501
002502
002503
002504
002505
002506

End of File -- Lines: 2506 Characters: 78752
```

```
================================================================================================
FILE: "LIBTK/UABC4.TEXT"
================================================================================================

000001  {INCLUDE FILE UABC4 -- IMPLEMENTATION OF UABC}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004                                  {TWindow-TDialogBox-TMenuBar-TFont}
000005
000006
000007  {changed 05/07/84 17:45 Fixed (hopefully) a bug in binary search of InAllMenusDo.}
000008
000009  METHODS OF TWindow;
000010
000011
000012      {$S SgABCini}
000013      FUNCTION  {TWindow.}CREATE{(object: TObject; heap: THeap; itsWmgrID: TWindowID; itsResizability: BOOLEAN)
000014                                 : TWindow};
000015          VAR pWindow:    WindowPtr;
000016              panels:     TList;
000017              info:       WindowInfo;
000018      BEGIN
000019          {$IFC fTrace}BP(7);{$ENDC}
000020          IF object = NIL THEN
000021              object := NewObject(heap, THISCLASS);
000022          SELF := TWindow(object);
000023
000024          GetWindInfo(WindowPtr(itsWmgrID), info);
000025
000026          WITH SELF DO
000027              BEGIN
000028              panelTree := NIL;
000029              dialogBox := NIL;
000030              selectPanel := NIL;
000031              undoSelPanel := NIL;
000032              clickPanel := NIL;
000033              undoClickPanel := NIL;
000034              wmgrID := itsWmgrID;
000035              isResizable := itsResizability;
000036              believeWmgr := info.visible;
000037              changes := 0;
000038              selectWindow := SELF ;
000039              undoSelWindow := NIL; {+SW+}
000040              lastCmd := NIL;
000041              parentBranch := NIL;
000042              pgSzOK := TRUE;       {client can explicitly set this to FALSE if bothered}
000043              pgRgOK := TRUE;       {client can explicitly set this to FALSE if does own page-ranging}
```

```
000044              panelToPrint := NIL;
000045              objectToFree := NIL; {+SW+}
000046              END;
000047
000048          panels := TList.CREATE(NIL, heap, 1);
000049          SELF.panels := panels;
000050
000051          IF itsWmgrID = 0 THEN
000052              SELF.SetInnerRect(zeroRect)
000053          ELSE
000054              BEGIN
000055              pWindow := POINTER(itsWmgrID);
000056              SELF.SetInnerRect(pWindow^.portRect);
000057              END;
000058
000059    {$H-} SELF.maxInnerSize := Point(FDiagRect(SELF.innerRect)); {$H+}
000060          {$IFC fTrace}EP;{$ENDC}
000061      END;
000062      {$S SgABCres}
000063
000064
000065      {$S SgABCini}
000066      PROCEDURE {TWindow.}Free;
000067      BEGIN
000068          {$IFC fTrace}BP(7);{$ENDC}
000069          Free(SELF.dialogBox);
000070          SELF.panels.Free;
000071          TArea.Free;
000072          {$IFC fTrace}EP;{$ENDC}
000073      END;
000074      {$S SgABCres}
000075
000076
000077      {$IFC fDebugMethods}
000078      {$S SgABCdbg}
000079      PROCEDURE {TWindow.}Fields{(PROCEDURE Field(nameAndType: S255))};
000080      BEGIN
000081          TArea.Fields(Field);
000082          Field('panels: TList');
000083          Field('panelTree: TArea');
000084          Field('dialogBox: TDialogBox');
000085          Field('selectPanel: TPanel');
000086          Field('undoSelPanel: TPanel');
000087          Field('clickPanel: TPanel');
000088          Field('undoClickPanel: TPanel');
000089          Field('selectWindow: TWindow');
000090          Field('undoSelWindow: TWindow');  {+SW+}
000091          Field('wmgrID: Ptr');
```

```
000092          Field('isResizable: BOOLEAN');
000093          Field('believeWmgr: BOOLEAN');
000094          Field('maxInnerSize: Point');
000095          Field('changes: LONGINT');
000096          Field('lastCmd: TCommand');
000097          Field(CONCAT('printerMetrics: RECORD paperRect: Rect; printRect: Rect;',
000098                       'res: Point; reserve: ARRAY[0..3] OF INTEGER END'));
000099          Field('pgSzOK: BOOLEAN');
000100          Field('pgRgOK: BOOLEAN');
000101          Field('panelToPrint: TPanel');
000102          Field('objectToFree: TObject');  {+SW+}
000103          Field('');
000104      END;
000105      {$S SgABCres}
000106      {$ENDC}
000107
000108
000109  {$S SgABCcld}
000110      PROCEDURE {TWindow.}AbortEvent;
000111      BEGIN
000112          {$IFC fTrace}BP(9);{$ENDC}
000113          {$IFC fTrace}EP;{$ENDC}
000114      END;
000115  {$S SgABCres}
000116
000117
000118  {$S SgABCpri}
000119      PROCEDURE {TWindow.}AcceptNewPrintingInfo{(document: TDocManager; prReserve: TPrReserve)};
000120       VAR  s:        TListScanner;
000121            panel:    TPanel;
000122      BEGIN
000123          {$IFC fTrace}BP(9);{$ENDC}
000124          SELF.selectPanel.selection.MarkChanged;
000125          IF document = clipboard THEN       {first, stuff the revised print record back in document}
000126              clipPrintPref := prReserve
000127          ELSE
000128              document.dataSegment.preludePtr^.printPref := prReserve;
000129          SELF.GetPrinterMetrics;
000130          s := SELF.panels.Scanner;
000131          WHILE s.Scan(panel) DO
000132              panel.currentView.ReactToPrinterChange;   {tell each view that printer style changed}
000133          {$IFC fTrace}EP;{$ENDC}
000134      END;
000135      {$S SgABCres}
000136
000137
000138      {$S sStartup}
000139      PROCEDURE {TWindow.}Activate; {assumes we are focused on the window already}
```

```
000140      BEGIN
000141          {$IFC fTrace}BP(7);{$ENDC}
000142          IF NOT SELF.believeWmgr THEN {is this needed????}
000143              SELF.Resize(FALSE);
000144
000145          SELF.Update(TRUE);                          {force update in case just opened from an icon}
000146          currentWindow := SELF;
000147
000148          {NOTE: currentDocument has already been set}
000149          activeWindowID := SELF.wmgrID;
000150          SELF.Refresh([rFrame], hDimToOn);
000151          IF currentDocument <> clipboard then
000152              SELF.ChkPrMismatch;
000153          SELF.PickStdCursor;
000154          clipboard.Inspect;
000155          IF SELF.dialogBox <> NIL THEN
000156              SELF.dialogBox.Appear;
000157          IF currentDocument.pendingNote <> 0 THEN
000158              BEGIN
000159              process.Note(currentDocument.pendingNote);
000160              currentDocument.pendingNote := 0;
000161              END;
000162
000163          {reset undo}
000164
000165          SELF.LoadMenuBar;
000166          menuBar.Draw;
000167
000168          SetPt(clickState.where, -MAXINT, -MAXINT);
000169          {$IFC fTrace}EP;{$ENDC}
000170      END;
000171      {$S SgABCres}
000172
000173
000174      {$S Override}
000175      PROCEDURE {TWindow.}BlankStationery;
000176          VAR panel:  TPanel;
000177              view:   TView;
000178      BEGIN
000179          {$IFC fTrace}BP(7);{$ENDC}
000180          panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
000181          view := panel.NewStatusView(NIL, zeroLRect);
000182          {$IFC fTrace}EP;{$ENDC}
000183      END;
000184      {$S SgABCres}
000185
000186
000187      {$S sCommand}
```

```
000188      FUNCTION  {TWindow.}CanDoCommand{(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN};
000189      BEGIN
000190          {$IFC fTrace}BP(6);{$ENDC}
000191          CanDoCommand := currentWindow.CanDoStdCommand(cmdNumber, checkIt);
000192          {$IFC fTrace}EP;{$ENDC}
000193      END;
000194
000195
000196      {$S sCommand}
000197      FUNCTION  {TWindow.}CanDoStdCommand{(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN};
000198          VAR previewMode:    TPreviewMode;
000199              couldPrint:     BOOLEAN;
000200              panelToUse:     TPanel;
000201      BEGIN
000202          {$IFC fTrace}BP(6);{$ENDC}
000203          CanDoStdCommand := FALSE;
000204
000205          couldPrint := (SELF.panelToPrint <> NIL);
000206          IF couldPrint THEN
000207              IF SELF.selectPanel.view.isPrintable THEN
000208                  panelToUse := SELF.selectPanel
000209              ELSE
000210                  panelToUse := SELF.panelToPrint;
000211
000212          IF couldPrint THEN
000213              previewMode := panelToUse.previewMode;
000214
000215          CASE cmdNumber OF
000216
000217            {File/Print}
000218
000219              uSetAllAside, uSetAside, uSetClipAside:
000220                  CanDoStdCommand := TRUE;
000221
000222              uPutAway, uRevertVersion:
000223                  CanDoStdCommand := clipboard.window <> SELF;
000224
000225              uSaveVersion:
000226                  CanDoStdCommand := (clipboard.window <> SELF) AND
000227                                     (currentDocument.files.shouldToolSave OR
000228                                      NOT currentDocument.openedAsTool);
000229
000230  {$IFC LibraryVersion <= 20}
000231              uPrFmt, uPrint:
000232                  CanDoStdCommand := onDesktop AND (SELF.dialogBox = NIL) AND couldPrint;
000233  {$ELSEC}
000234              uPrFmt, uPrint, uPrintAsIs:
000235                  CanDoStdCommand := onDesktop AND (SELF.dialogBox = NIL) AND couldPrint;
```

```
000236  {$ENDC}
000237
000238              uPrMonitor:
000239                  CanDoStdCommand := onDesktop AND (SELF.dialogBox = NIL);        {**temporary**}
000240
000241            {Edit}
000242
000243              uUndoLast:
000244                  IF SELF.lastCmd = NIL THEN
000245                      CanDoStdCommand := FALSE
000246                  ELSE
000247                      CanDoStdCommand := SELF.lastCmd.undoable;
000248
000249            {Page Layout}
000250
000251              uPrvwMargins, uPrvwBreaks, uPrvwOff, uAddColumnStrip, uAddRowStrip:
000252                  IF couldPrint THEN
000253                      BEGIN
000254                      CanDoStdCommand := TRUE;  {or they wouldnt've been in the phrase file}
000255                      CASE cmdNumber OF
000256                          uPrvwMargins:
000257                              checkIt := previewMode = mPrvwMargins;
000258                          uPrvwBreaks:
000259                              checkIt := previewMode = mPrvwBreaks;
000260                          uPrvwOff:
000261                              checkIt := previewMode = mPrvwOff;
000262                          END;
000263                      END;
000264
000265              uDesignPages:
000266                  IF couldPrint THEN
000267                      BEGIN
000268                      CanDoStdCommand := (SELF.dialogBox = NIL);
000269                      checkIt := (SELF.dialogBox = panelToUse.view.printManager.layoutDialogBox)
000270                                      AND (SELF.dialogBox <> NIL);
000271                      END;
000272
000273              uSetHorzBreak, uSetVertBreak, uClearBreaks:
000274                  CanDoStdCommand := SELF.clickPanel.view.isPrintable;
000275
000276              uShowFullSize, uReduce70Pct, uReduceToFit:
000277                  CanDoStdCommand := fExperimenting;                                  {**temporary**}
000278
000279              uRiseVertically, uRiseHorizontally:
000280                  IF couldPrint THEN
000281                      BEGIN
000282                      CanDoStdCommand := TRUE;
000283                      checkIt := panelToUse.view.printManager.pageRiseDirection =
```

```
000284                                                          VHSelect(cmdNumber = uRiseHorizontally);
000285                  END
000286              ELSE
000287                  CanDoStdCommand := FALSE;
000288
000289          {$IFC fDbgABC}
000290          {Debug}
000291
000292            uReportEvents, uCountHeap, uCheckIndices,
000293              uExperimenting, uDumpGlobals, uDumpPrelude,
000294              uMainScramble, uDocScramble:
000295                BEGIN
000296                CanDoStdCommand := TRUE;
000297                CASE cmdNumber OF
000298                    uReportEvents:
000299                        checkIt := eventDebug;
000300                    uCountHeap:
000301                        checkIt := fCountHeap;
000302                    uCheckIndices:
000303                        checkIt := fCheckIndices;
000304                    uExperimenting:
000305                        checkIt := fExperimenting;
000306                    uMainScramble:
000307                        checkIt := THz(mainHeap)^.fScramble;
000308                    uDocScramble:
000309                        IF currentDocument <> NIL THEN
000310                            checkIt := THz(currentDocument.docHeap)^.fScramble
000311                        ELSE
000312                            CanDoStdCommand := FALSE;
000313                    END;
000314                END;
000315
000316            uReptGarbage, uFreeGarbage:                                 {Debug}
000317                CanDoStdCommand := clipboard.window <> SELF;
000318          {$ENDC}
000319
000320            END;
000321        {$IFC fTrace}EP;{$ENDC}
000322    END;
000323  {$S SgABCres}
000324
000325
000326    {$S sStartup}
000327    PROCEDURE {TWindow.}ChkPrMismatch;
000328        VAR styleDidChange: BOOLEAN;
000329            prPrfAlias:     TPrPrfAlias;
000330            s:              TListScanner;
000331            panel:          TPanel;
```

```
000332              error:          INTEGER;
000333              document:       TDocManager;
000334      BEGIN
000335          {$IFC fTrace}BP(7);{$ENDC}
000336          IF currentDocument <> NIL THEN
000337              document := currentDocument
000338          ELSE
000339              document := boundDocument;
000340          IF document = clipboard THEN
000341              prPrfAlias.reserve := clipPrintPref
000342          ELSE
000343              prPrfAlias.reserve := document.dataSegment.preludePtr^.PrintPref;
000344  {$IFC libraryVersion <= 20}         { P E P S I }
000345          IF FPrArbRqd(prPrfAlias.prPrf) THEN
000346              BEGIN
000347              PrArbDlg(error, prPrfAlias.prPrf, styleDidChange);
000348  {$ELSEC}                            { S P R I N G }
000349          IF NOT fPrPrfValid(prPrfAlias.prPrf) THEN
000350              BEGIN
000351              PrPrfDlg(prPrfAlias.prPrf, styleDidChange, NOT SELF.pgSzOK);
000352  {$ENDC}
000353
000354              IF styleDidChange THEN
000355                  SELF.AcceptNewPrintingInfo(document, prPrfAlias.reserve);
000356              END;
000357          {?? Do we need to worry about refreshing the window when needed?}
000358          {$IFC fTrace}EP;{$ENDC}
000359      END;
000360
000361
000362      {$S sCommand}
000363      PROCEDURE {TWindow.}CommitLast;
000364          VAR lastCmd:    TCommand;
000365              lastView:   TView; {+SW+}
000366              selection:  TSelection; {+SW+}
000367      BEGIN
000368          {$IFC fTrace}BP(7);{$ENDC}
000369          IF SELF <> currentWindow THEN
000370              currentWindow.CommitLast
000371          ELSE
000372              BEGIN
000373              lastCmd := SELF.lastCmd;
000374              IF lastCmd <> NIL THEN
000375                  BEGIN
000376                  IF lastCmd.doing THEN
000377                      lastCmd.Commit;
000378  (*****
000379                  IF lastCmd.image <> NIL THEN
```

```
000380                      BEGIN
000381                      lastView := lastCmd.image.view;
000382                      selection := lastView.panel.undoSelection.FreedAndReplacedBy(lastView.NoSelection);
000383                      END;
000384  *****)
000385                  lastCmd.Free;
000386                  SELF.lastCmd := NIL;
000387                  END;
000388              END;
000389          {$IFC fTrace}EP;{$ENDC}
000390      END;
000391
000392
000393      {$S sStartup}
000394      FUNCTION  {TWindow.}CursorFeedback{: TCursorNumber};
000395          VAR s:              TListScanner;
000396              panel:          TPanel;
000397              cursorNumber:   TCursorNumber;
000398              mousePt:        Point;
000399      BEGIN
000400          {$IFC fTrace}BP(3);{$ENDC}
000401          PushFocus;
000402          SELF.Focus;
000403          cursorNumber := noCursor;
000404          GetMouse(mousePt);
000405          IF RectHasPt(SELF.innerRect, mousePt) THEN
000406              IF SELF.isResizable AND fGrowHit(mousePt) THEN
000407                  cursorNumber := arrowCursor
000408              ELSE
000409                  BEGIN
000410                  s := SELF.panels.Scanner;
000411                  WHILE s.Scan(panel) DO
000412                      BEGIN
000413                      cursorNumber := panel.CursorAt(mousePt);
000414                      IF cursorNumber <> noCursor THEN
000415                          s.Done;
000416                      END;
000417                  IF cursorNumber = noCursor THEN
000418                      cursorNumber := arrowCursor;
000419                  END;
000420          PopFocus;
000421          CursorFeedback := cursorNumber;
000422          {$IFC fTrace}EP;{$ENDC}
000423      END;
000424
000425
000426      {$S SgABCcld}
000427      PROCEDURE {TWindow.}Deactivate; {assumes we are focused on the window already}
```

```
000428      BEGIN
000429          {$IFC fTrace}BP(7);{$ENDC}
000430  (******************** these lines are needed for the Extra Window feature *)
000431          IF currentWindow <> SELF THEN
000432              BEGIN
000433              GiveControl(event); {This must be last}
000434              {$IFC fTrace}EP;{$ENDC}
000435              EXIT(Deactivate);
000436              END;
000437  (* ********************)
000438          SELF.CommitLast;
000439
000440          IF SELF.dialogBox <> NIL THEN
000441              SELF.dialogBox.Disappear;
000442
000443          activeWindowID := 0; {must precede StashPicture and Refresh so scroll bars are white}
000444          SELF.Refresh([rFrame], hOnToDim); {do first to give user feedback}
000445          SELF.StashPicture(hOfftoDim);
000446          IF (SELF.wmgrId <> ORD(scrapFolder)) AND (event.fromProcess <> myProcessID) THEN
000447              clipboard.Publicize;
000448
000449          focusArea := NIL;
000450
000451          IF NOT inBackground THEN
000452              currentDocument.Deactivate;
000453
000454          GiveControl(event); {This must be last}
000455          {$IFC fTrace}EP;{$ENDC}
000456      END;
000457      {$S SgABCres}
000458
000459
000460      {$S sCommand}
000461      PROCEDURE {TWindow.}DoCommand{(cmdNumber: TCmdNumber)};
000462          VAR command: TCommand;
000463      BEGIN
000464          {$IFC fTrace}BP(7);{$ENDC}
000465          IF cmdNumber <> 0 THEN
000466              BEGIN
000467              IF cmdNumber = uUndoLast THEN
000468                  SELF.UndoLast
000469              ELSE
000470                  BEGIN
000471                  command := SELF.selectPanel.selection.NewCommand(cmdNumber);
000472                  IF command <> NIL THEN      {NOTE: If NewCommand Frees SELF (this window), it MUST return NIL}
000473                      SELF.PerformCommand(command);
000474                  END;
000475              process.RememberCommand(cmdNumber);
```

```
000476              END;
000477          menuBar.EndCmd;
000478          {$IFC fTrace}EP;{$ENDC}
000479      END;
000480
000481
000482      {$S sClick}
000483      FUNCTION  {TWindow.}DownAt{(mousePt: Point): BOOLEAN};
000484          VAR s:      TListScanner;
000485              panel:  TPanel;
000486              b:      BOOLEAN;
000487      BEGIN
000488          {$IFC fTrace}BP(7);{$ENDC}
000489          b := FALSE;
000490          IF RectHasPt(SELF.innerRect, mousePt) THEN
000491              BEGIN
000492              IF SELF.isResizable THEN
000493                  IF fGrowHit(mousePt) THEN
000494                      BEGIN
000495                      SELF.DownInSizeBox(mousePt);
000496                      b := TRUE;
000497                      process.RememberCommand(uResizeWindow);
000498                      END;
000499              IF NOT b THEN
000500                  BEGIN
000501                  b := TRUE;
000502                  s := SELF.panels.Scanner;
000503                  WHILE s.Scan(panel) DO
000504                      IF panel.DownAt(mousePt) THEN
000505                          s.Done;
000506                  END;
000507              END;
000508          DownAt := b;
000509          {$IFC fTrace}EP;{$ENDC}
000510      END;
000511
000512
000513      {$S sClick}
000514      PROCEDURE {TWindow.}DownEventAt{(mousePt: Point)};
000515      VAR clickNeighborhood:  Rect;
000516      BEGIN
000517          {$IFC fTrace}BP(7);{$ENDC}
000518          SELF.Update(TRUE);  {In case an alert box was dismissed by the click}
000519
000520              { given that previous click was at (0,0), clickNeighborhood is a rectangle in which
000521                  this click must fall for it to have a chance at being a double click }
000522          SetRect(clickNeighborhood, -9, -6, 9, 6);   { clickNeighborhood should be a method call;
000523                                                          how much flexibility is needed???? }
```

```
000524
000525          IF ((event.when - clickState.when) < clickDelay) AND
000526             (RectHasPt(clickNeighborhood, Point(FPtMinusPt(event.where, clickState.where)))) THEN
000527              clickState.clickCount := Min(clickState.clickCount + 1, 3)
000528          ELSE
000529              BEGIN
000530              clickState.clickCount := 1;
000531              clickState.fShift := event.shiftKey;
000532              clickState.fOption := event.codeKey;
000533              clickState.fApple := event.appleKey;
000534              END;
000535          clickState.when := event.when;
000536          clickState.where := event.where;
000537          IF SELF.DownAt(mousePt) THEN;
000538          {$IFC fTrace}EP;{$ENDC}
000539      END;
000540
000541
000542      {$S sClick}
000543      PROCEDURE {TWindow.}DownInSizeBox{(mousePt: Point)};
000544          VAR oldRect:        Rect;
000545              fullRect:       Rect;   {includes title tab}
000546              minExtent:      Point;
000547              minBotRight:    Point;
000548              maxBotRight:    Point;
000549              savePort:       GrafPtr;
000550              newBotRight:    Point;
000551      BEGIN
000552          {$IFC fTrace}BP(7);{$ENDC}
000553          oldRect := SELF.innerRect;
000554
000555          SELF.GetMinExtent(minExtent, TRUE);
000556          minBotRight := Point(FPtPlusPt(oldRect.topLeft, minExtent));
000557
000558          LocalToGlobal(minBotRight);
000559          LocalToGlobal(mousePt);
000560          LocalToGlobal(oldRect.topLeft);
000561          LocalToGlobal(oldRect.botRight);
000562          maxBotRight := Point(FPtMaxPt(minBotRight, screenBits.bounds.botRight));
000563
000564          fullRect := oldRect;
000565          fullRect.top := fullRect.top - dvSBox; {allow for title tab}
000566
000567          GetPort(savePort);
000568          SetPort(deskPort);
000569          ResizeFeedback(mousePt, minBotRight, maxBotRight, fullRect, dvSBox, dhSBox, dvSBox, newBotRight);
000570          SetPort(savePort);
000571
```

```
000572          SELF.ResizeTo(Point(FPtMinusPt(newBotRight, oldRect.topLeft)));
000573          {$IFC fTrace}EP;{$ENDC}
000574      END;
000575
000576
000577      {$S sFilter}  {+SW+}
000578      PROCEDURE {TWindow.}EachActualPart{(PROCEDURE DoToObject(filteredObj: TObject))};
000579          VAR n:          INTEGER;
000580              cmdWindow:  TWindow;
000581      BEGIN
000582          {$IFC fTrace}BP(11);{$ENDC}
000583          {$IFC fDbgABC}
000584          IF SELF = currentWindow.dialogBox THEN
000585              cmdWindow := currentWindow
000586          ELSE
000587              cmdWindow := SELF;
000588
000589          IF cmdWindow.lastCmd = NIL THEN
000590              n := 0
000591          ELSE
000592              n := cmdWindow.lastCmd.cmdNumber;
000593          ABCBreak('A View or Window tried to filter but did not implement EachActualPart: lastCmd =', n);
000594          {$ENDC}
000595          {$IFC fTrace}EP;{$ENDC}
000596      END;
000597      {$S SgABCres}
000598
000599
000600      {$S sFilter}
000601      PROCEDURE {TWindow.}EachVirtualPart{(PROCEDURE DoToObject(filteredObj: TObject))};
000602      BEGIN
000603          {$IFC fTrace}BP(11);{$ENDC}
000604          SELF.FilterDispatch(NIL, NIL, DoToObject);
000605          {$IFC fTrace}EP;{$ENDC}
000606      END;
000607
000608
000609      {$S sFilter}
000610      PROCEDURE {TWindow.}FilterAndDo{(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject))};
000611      BEGIN
000612          {$IFC fTrace}BP(11);{$ENDC}
000613          SELF.FilterDispatch(actualObj, NIL, DoToObject);
000614          {$IFC fTrace}EP;{$ENDC}
000615      END;
000616
000617
000618      {$S sFilter}
000619      PROCEDURE {TWindow.}FilterDispatch{(actualObj: TObject; image: TImage;
```

```
000620                                          PROCEDURE DoToObject(filteredObj: TObject))};
000621      VAR filterCommand:  TCommand;
000622          filtering:      BOOLEAN;
000623          cmdWindow:      TWindow;
000624      BEGIN
000625          {$IFC fTrace}BP(11);{$ENDC}
000626          cmdWindow := SELF;
000627          IF currentWindow <> NIL THEN
000628              IF SELF = currentWindow.dialogBox THEN
000629                  cmdWindow := currentWindow;
000630
000631          filterCommand := cmdWindow.lastCmd;
000632
000633          filtering := FALSE;
000634          IF filterCommand <> NIL THEN
000635              IF filterCommand.doing THEN
000636                  IF filterCommand.image <> NIL THEN
000637                      filtering := filterCommand.image.SeesSameAs(image);
000638
000639          IF filtering THEN
000640              IF actualObj <> NIL THEN
000641                  filterCommand.FilterAndDo(actualObj, DoToObject)
000642              ELSE
000643                  filterCommand.EachVirtualPart(DoToObject)
000644          ELSE
000645          IF actualObj <> NIL THEN
000646              DoToObject(actualObj)
000647          ELSE
000648          IF image <> NIL THEN
000649              image.EachActualPart(DoToObject)
000650          ELSE
000651              SELF.EachActualPart(DoToObject);
000652          {$IFC fTrace}EP;{$ENDC}
000653      END;
000654
000655
000656  {$S sStartup}
000657      PROCEDURE {TWindow.}Focus;
000658      BEGIN
000659          {$IFC fTrace}BP(6);{$ENDC}
000660          SetPort(POINTER(SELF.wmgrID));
000661          SetOrigin(0, 0);
000662          ClipRect(thePort^.portRect);
000663          IF useAltVisRgn THEN
000664              focusRgn := altVisRgn       {Instigated by TWindow.StashPicture or TClipboard.Publicize}
000665          ELSE
000666              focusRgn := thePort^.visRgn;
000667          focusArea := SELF;
```

```
000668          {$IFC fTrace}EP;{$ENDC}
000669      END;
000670  {$S SgABCres}
000671
000672
000673  {$S sStartup}
000674      PROCEDURE {TWindow.}Frame;
000675          VAR growRect: Rect;
000676      BEGIN
000677          {$IFC fTrace}BP(6);{$ENDC}
000678          IF SELF.isResizable THEN
000679              BEGIN
000680              GetGrowRect(growRect);
000681              IF RectIsVisible(growRect) THEN
000682                  IF SELF.IsActive THEN
000683                      PaintGrow
000684                  ELSE
000685                      FillRect(growRect, white);
000686              END;
000687          {$IFC fTrace}EP;{$ENDC}
000688      END;
000689  {$S SgABCres}
000690
000691
000692      {$S sCldInit}
000693      PROCEDURE {TWindow.}GetPrinterMetrics;
000694           VAR  prPrfAlias:   TPrPrfAlias;
000695                prInfo:       TPrInfo;
000696                tkDevice:     INTEGER;
000697                document:     TDocManager;
000698      BEGIN
000699          {$IFC fTrace}BP(9);{$ENDC}
000700          IF currentDocument <> NIL THEN
000701              document := currentDocument
000702          ELSE
000703              document := boundDocument;
000704          IF document = clipboard THEN
000705              prPrfAlias.reserve := clipPrintPref
000706          ELSE
000707              prPrfAlias.reserve := document.dataSegment.preludePtr^.printPref;
000708  {$IFC libraryVersion <= 20}         { P E P S I }
000709          PrMetrics(prPrfAlias.prPrf, prInfo);
000710  {$ELSEC}                            { S P R I N G }
000711          prInfo := prPrfAlias.prPrf.prInfo;  {this looks odd, but the prPrf is of type prRec really}
000712  {$ENDC}
000713          WITH SELF.printerMetrics, prInfo DO
000714              BEGIN
000715              printRect := rPrintable;
```

```
000716              paperRect := rPaper;
000717              END;
000718          SELF.printerMetrics.res.h := prInfo.hRes;
000719          SELF.printerMetrics.res.v := prInfo.vRes;
000720          {$IFC fTrace}EP;{$ENDC}
000721      END;
000722      {$S SgABCres}
000723
000724
000725      {$S sStartup}
000726      PROCEDURE {TWindow.}GetMinExtent{(VAR minExtent: Point; windowIsResizingIt: BOOLEAN)};
000727      BEGIN
000728          {$IFC fTrace}BP(9);{$ENDC}
000729          SELF.panelTree.GetMinExtent(minExtent, windowIsResizingIt);
000730          {$IFC fTrace}EP;{$ENDC}
000731      END;
000732      {$S SgABCres}
000733
000734
000735      {$S sRes}
000736      PROCEDURE {TWindow.}GetTitle{(VAR title: S255)};
000737          VAR kludge: Str255;
000738      BEGIN
000739          {$IFC fTrace}BP(7);{$ENDC}
000740          GetFldrTitle(POINTER(SELF.wmgrID), kludge);
000741          title := kludge;
000742          {$IFC fTrace}EP;{$ENDC}
000743      END;
000744
000745
000746      {$S sStartup}
000747      PROCEDURE {TWindow.}Highlight{(highTransit: THighTransit)};
000748          PROCEDURE HilitePanel(obj: TObject);
000749          BEGIN
000750              TPanel(obj).Highlight(TPanel(obj).selection, highTransit);
000751          END;
000752      BEGIN
000753          {$IFC fTrace}BP(7);{$ENDC}
000754          SELF.panels.Each(HilitePanel);
000755          {$IFC fTrace}EP;{$ENDC}
000756      END;
000757  {$S SgABCres}
000758
000759
000760      {$S sRes}
000761      PROCEDURE {TWindow.}IdleBegin{(centiSeconds: LONGINT)};
000762      BEGIN
000763          {$IFC fTrace}BP(7);{$ENDC}
```

```
000764          LetOthersRun;
000765          {$IFC fTrace}EP;{$ENDC}
000766      END;
000767
000768
000769      {$S sRes}
000770      PROCEDURE {TWindow.}IdleContinue{(centiSeconds: LONGINT)};
000771      BEGIN
000772          {$IFC fTrace}BP(5);{$ENDC}
000773          IF SELF.IsActive THEN
000774              process.TrackCursor;
000775          LetOthersRun;
000776          {$IFC fTrace}EP;{$ENDC}
000777      END;
000778
000779
000780      {$S sRes}
000781      PROCEDURE {TWindow.}IdleEnd{(centiSeconds: LONGINT)};
000782      BEGIN
000783          {$IFC fTrace}BP(7);{$ENDC}
000784          {$IFC fTrace}EP;{$ENDC}
000785      END;
000786
000787
000788      {$S sStartup}
000789      FUNCTION  {TWindow.}IsActive{: BOOLEAN};
000790      BEGIN
000791          {$IFC fTrace}BP(3);{$ENDC}
000792          IF activeWindowID = 0 THEN  {nothing is active}
000793              IsActive := FALSE
000794          ELSE IF currentWindow = NIL THEN
000795              BEGIN
000796              IsActive := FALSE;
000797              {$IFC fDbgABC}
000798              Writeln(CHR(7), '********************');
000799              Writeln('In TWindow.IsActive, activeWindowID <> 0 AND currentWindow = NIL');
000800              Writeln('activeWindowID=', activeWindowID:1, '   currentWindow=', ORD(currentWindow):1);
000801              Writeln('********************');
000802              {$ENDC}
000803              END
000804          ELSE
000805              IsActive := (SELF.wmgrID = activeWindowID) OR (SELF.wmgrID = ORD(dialogFolder));
000806          {$IFC fTrace}EP;{$ENDC}
000807      END;
000808
000809
000810      {$S sStartup}
000811      FUNCTION  {TWindow.}IsVisible{: BOOLEAN};
```

```
000812          VAR info:   WindowInfo;
000813      BEGIN
000814          {$IFC fTrace}BP(3);{$ENDC}
000815          GetWindInfo(WindowPtr(SELF.wmgrID), info);
000816          IsVisible := info.visible;
000817          {$IFC fTrace}EP;{$ENDC}
000818      END;
000819      {$S SgABCres}
000820
000821
000822      {$S sStartup}
000823      PROCEDURE {TWindow.}LoadMenuBar;
000824          VAR i:              INTEGER;
000825              menuID:         INTEGER;
000826              inClipboard:    BOOLEAN;
000827      BEGIN
000828          {$IFC fTrace}BP(7);{$ENDC}
000829          inClipboard := activeWindowID = ORD(scrapFolder);
000830
000831          FOR i := 1 TO menuBar.numMenus DO
000832              BEGIN
000833              menuID := wmgrMenus[i].menuID;
000834              IF SELF.WantMenu(menuID, inClipboard) THEN
000835                  menuBar.Insert(menuID, 0);
000836              END;
000837          {$IFC fTrace}EP;{$ENDC}
000838      END;
000839      {$S SgABCres}
000840
000841
000842      {$S sRes}
000843      PROCEDURE {TWindow.}MenuEventAt{(mousePt: Point)};
000844          VAR cmdNumber: TCmdNumber;
000845      BEGIN
000846          {$IFC fTrace}BP(7);{$ENDC}
000847          SELF.SetupMenus;
000848          cmdNumber := menuBar.DownAt(mousePt);
000849          IF SELF.selectPanel = NIL THEN
000850              {$IFC fDbgABC} ABCBreak('ObeyTheEvent: selectPanel=NIL', 0) {$ENDC}
000851          ELSE
000852              SELF.DoCommand(cmdNumber);
000853          {$IFC fTrace}EP;{$ENDC}
000854      END;
000855
000856
000857  {$IFC LibraryVersion > 20}
000858  {$S SgABCcld}
000859      PROCEDURE {TWindow.}NameToPrefix(VAR error, offset: INTEGER; VAR name, prefix: TFilePath);
```

```
000860      BEGIN
000861          {$IFC fTrace}BP(7);{$ENDC}
000862          NameToPrefix(error, offset, WindowPtr(SELF.wmgrID), Pathname(name), Pathname(prefix));
000863          {$IFC fTrace}EP;{$ENDC}
000864      END;
000865  {$S SgABCres}
000866  {$ENDC}
000867
000868
000869      {$S sCommand}
000870      FUNCTION  {TWindow.}NewCommand{(cmdNumber: TCmdNumber): TCommand};
000871      BEGIN
000872          {$IFC fTrace}BP(7);{$ENDC}
000873          NewCommand := currentWindow.NewStdCommand(cmdNumber);
000874          {$IFC fTrace}EP;{$ENDC}
000875      END;
000876
000877
000878      {$S sCommand}
000879      FUNCTION  {TWindow.}NewStdCommand{(cmdNumber: TCmdNumber): TCommand};
000880      VAR document:       TDocManager;
000881          didStyleChange: BOOLEAN;
000882  {$IFC LibraryVersion <= 20}
000883          prPrf:          TPrPrf;
000884  {$ENDC}
000885          prPrfAlias:     TprPrfAlias;
000886          shouldPrint:    BOOLEAN;
000887          error:          INTEGER;
000888          str:            S255;
000889          permCmd:        BOOLEAN;    { TRUE iff the command is a permanent one }
000890          command:        TCommand;
000891          s:              TListScanner;
000892          panel:          TPanel;
000893          zoomNum:        Point;
000894          zoomDen:        Point;
000895          selectPanel:    TPanel;
000896          clickPanel:     TPanel;
000897          selection:      TSelection;
000898          vhs:            VHSelect;
000899          andContinue:    BOOLEAN;
000900          excessBytes:    INTEGER;
000901          printManager:   TPrintManager;
000902          panelToUse:     TPanel;
000903
000904          FUNCTION  RevertConfirmed: BOOLEAN;
000905              VAR s:          TParamAlert;
000906                  ph:         INTEGER;
000907  {$IFC LibraryVersion <= 20}
```

```
000908                  info:       fs_info;
000909  {$ELSEC}
000910                  info:       Q_Info;
000911  {$ENDC}
000912                  osErr:      INTEGER;
000913                  pPath:      ^Pathname;
000914                  osDT:       LONGINT;
000915          BEGIN
000916              RevertConfirmed := FALSE;
000917
000918              IF SELF.changes = 0 THEN
000919                  process.Note(phUnchanged)
000920
000921              ELSE
000922                  BEGIN
000923                  IF document.files.saveExists THEN
000924                      BEGIN
000925                      pPath := @document.files.volumePrefix;
000926  {$IFC LibraryVersion <= 20}
000927                      Lookup(osErr, pPath^, info);
000928  {$ELSEC}
000929                      Quick_Lookup(osErr, pPath^, info);
000930  {$ENDC}
000931                      IF osErr <= 0 THEN
000932                          osDT := info.DTM
000933                      ELSE
000934                          osDT := -1;
000935                      {$IFC LibraryVersion < 13}
000936                      DTAlert(osDT, s);
000937                      {$ELSEC}
000938                      DTAlert(alerts, osDT, s);
000939                      {$ENDC}
000940                      process.ArgAlert(1, s);
000941                      ph := phRevert;
000942                      END
000943                  ELSE
000944                      ph := phRevBlank;
000945
000946                  IF process.Caution(ph) THEN
000947                      RevertConfirmed := TRUE;
000948                  END;
000949          END;
000950      BEGIN
000951          {$IFC fTrace}BP(7);{$ENDC}
000952          document := currentDocument;
000953          {$IFC fDbgABC}
000954          IF SELF.wmgrID <> document.window.wmgrID THEN
000955              ABCbreak('In TWindow.NewStdCommand: SELF.wmgrID <> document.window.wmgrID; document=',
```

```
000956                      ORD(document));
000957          {$ENDC}
000958          selectPanel := SELF.selectPanel;
000959          clickPanel := SELF.clickPanel;
000960          selection := selectPanel.selection;
000961
000962          IF selectPanel.view.isPrintable THEN
000963              panelToUse := selectPanel
000964          ELSE
000965              panelToUse := SELF.panelToPrint;
000966          error := 0;
000967          NewStdCommand := NIL; {the default return value}
000968          permCmd := FALSE; {if set to TRUE, make a permanent command object}
000969          allowAbort := TRUE; {??? should we assume this ???}
000970
000971          CASE cmdNumber OF
000972
000973             {File/Print Menu}
000974              uSetAllAside:
000975                  BEGIN
000976                  SELF.CommitLast;
000977                  DoFilingCmd(cmdClosAll);
000978                  permCmd := TRUE;
000979                  END;
000980              uSetAside, uSetClipAside:
000981                  BEGIN
000982                  SELF.CommitLast;
000983                  DoFilingCmd(cmdClose);
000984                  permCmd := TRUE;
000985                  END;
000986              uPutAway, uSaveVersion: {must be the active window to do this}
000987                  BEGIN
000988                  andContinue := cmdNumber = uSaveVersion;
000989
000990                  SELF.CommitLast;
000991                  IF andContinue THEN
000992                      excessBytes := docExcess
000993                  ELSE
000994                      excessBytes := 0;
000995                  document.ConserveMemory(excessBytes, TRUE {do GC});
000996
000997                  IF (document.window.changes <> 0) AND
000998                     (document.files.shouldToolSave OR NOT document.openedAsTool) THEN
000999                      BEGIN
001000                      process.BeginWait(phSaving);
001001                      document.SaveVersion(error, document.files.volumePrefix, andContinue);
001002                      process.EndWait;
001003                      END
```

```
001004                  ELSE IF andContinue THEN
001005                      process.Note(phUnchanged);
001006                  {shouldn't we put up a message on Save & Put Away even if document is unchanged???}
001007
001008                  IF (error <= 0) AND NOT andContinue THEN      {*** some cases worse! ***}
001009                      BEGIN
001010                      TellFiler(error, docClosd, docPutBack, POINTER(activeWindowID));
001011                      IF error > 0 THEN
001012                          BEGIN
001013                          ABCBreak('TellFiler', error);
001014                          error := 0;
001015                          END;
001016                      closedDocument := document;
001017                      closedBySuspend := FALSE;
001018                      END;
001019
001020                  {do something if there was an error}
001021                  IF error = erAborted THEN
001022                      process.Stop(phTerminated)
001023                  ELSE IF error > 0 THEN
001024                      process.Stop(phCantSave);
001025
001026                  error := 0; {we already put up the alert}
001027                  permCmd := TRUE;
001028                  END;
001029              uRevertVersion:
001030                  IF RevertConfirmed THEN
001031                      BEGIN
001032                      document.RevertVersion(error, activeWindowID);
001033                      {do something if there was an error}
001034                      IF error = erAborted THEN
001035                          process.Stop(phTerminated)
001036                      ELSE IF error > 0 THEN
001037                          BEGIN
001038                          process.Stop(phCantRevert);
001039                          process.Complete(FALSE); {nothing else to do: we unbound out data segments}
001040                          END;
001041
001042                      error := 0; {we already put up the alert}
001043                      permCmd := TRUE; {no need to CommitLast}
001044                      END;      {long-standing commented-out code now out}
001045              uPrFmt:
001046                  BEGIN
001047                  IF document = clipboard THEN
001048                      prPrfAlias.reserve := clipPrintPref
001049                  ELSE
001050                      prPrfAlias.reserve := document.dataSegment.preludePtr^.printPref;
001051                  PushFocus;
```

```
001052  {$IFC libraryVersion <= 20}         { P E P S I }
001053                  PrPrfDlg(error, prPrfAlias.prPrf, didStyleChange);
001054  {$ELSEC}                            { S P R I N G }
001055                  PrPrfDlg(prPrfAlias.prPrf, didStyleChange, NOT SELF.pgSzOK);
001056  {$ENDC}
001057                  PopFocus;
001058                  IF didStyleChange THEN
001059                      BEGIN
001060                      SELF.AcceptNewPrintingInfo(document, prPrfAlias.reserve);
001061                      permCmd := TRUE;
001062                      END;
001063                  END;
001064
001065              uPrint:
001066                  SELF.Print(panelToUse, NOT SELF.pgRgOK, FALSE {put up dialog} );
001067
001068              uPrintAsIs:
001069                  SELF.Print(panelToUse, TRUE {suppress page range}, TRUE {suppress dialog});
001070
001071              uPrMonitor:
001072                  BEGIN
001073                  PushFocus;
001074  {$IFC libraryVersion <= 20}         { P E P S I }
001075                  PrBgdDlg(error, TRUE);
001076  {$ELSEC}                            { S P R I N G }
001077                  PrBgdDlg;
001078  {$ENDC}
001079                  PopFocus;
001080                  END;
001081
001082             {Zooming & previewing pages}      {some or all of these must become command objects}
001083              uPrvwMargins:
001084                  panelToUse.Preview(mPrvwMargins);
001085
001086              uPrvwBreaks:
001087                  panelToUse.Preview(mPrvwBreaks);
001088
001089              uPrvwOff:
001090                  panelToUse.Preview(mPrvwOff);
001091
001092              uDesignPages:
001093                  BEGIN
001094                  printManager := panelToUse.view.printManager;
001095                  IF printManager <> NIL THEN
001096                      printManager.EnterPageEdit;
001097                  END;
001098
001099              uRiseVertically,
```

```
001100                  uRiseHorizontally:
001101                      BEGIN
001102                      IF cmdNumber = uRiseVertically THEN
001103                          panelToUse.view.printManager.pageRiseDirection := v
001104                      ELSE
001105                          panelToUse.view.printManager.pageRiseDirection := h;
001106                      IF panelToUse.previewMode = mPrvwMargins THEN
001107                          panelToUse.Invalidate;
001108                      END;
001109
001110              uAddColumnStrip,
001111                  uAddRowStrip:
001112                      BEGIN
001113                      IF cmdNumber = uAddColumnStrip THEN
001114                          vhs := v
001115                      ELSE
001116                          vhs := h;
001117                      panelToUse.currentView.AddStripOfPages(vhs);
001118                      END;
001119
001120              uShowFullSize:
001121                  BEGIN
001122                  SetPt(zoomNum, 1, 1);
001123                  selectPanel.SetZoomFactor(zoomNum, zoomNum);   {++ should this be panelToUse?? ++}
001124                  selectPanel.Invalidate;
001125                  END;
001126
001127              uReduce70Pct:
001128                  BEGIN
001129                  WITH selectPanel.zoomFactor DO
001130                      IF numerator.h = 1 THEN
001131                          BEGIN
001132                              zoomNum.h := 7;
001133                              zoomDen.h := denominator.h * 10;
001134                              zoomNum.v := 7;
001135                              zoomDen.v := denominator.v * 10;
001136                          END
001137                      ELSE {numerator not 1, must be 7}
001138                          BEGIN
001139                              zoomNum.h := 1;
001140                              zoomDen.h := denominator.h DIV 5;
001141                              zoomNum.v := 1;
001142                              zoomDen.v := denominator.v DIV 5;
001143                          END;
001144                  selectPanel.SetZoomFactor(zoomNum, zoomDen);
001145                  selectPanel.Invalidate;
001146                  END;
001147
```

```
001148              uReduceToFit:
001149                  {?} {can't do it now--how to express?} ;
001150
001151              uSetHorzBreak:
001152                  IF clickPanel.view.isPrintable THEN
001153                      BEGIN
001154                      clickPanel.view.printManager.SetBreak(h, clickPanel.view.clickLPt.v, FALSE);
001155                      clickPanel.currentView.ReDoBreaks;
001156                      clickPanel.Invalidate;
001157                      END;
001158
001159              uSetVertBreak:
001160                  IF clickPanel.view.isPrintable THEN
001161                      BEGIN
001162                      clickPanel.view.printManager.SetBreak(v, clickPanel.view.clickLPt.h, FALSE);
001163                      clickPanel.currentView.ReDoBreaks;
001164                      clickPanel.Invalidate;  {later, do a more selective inval}
001165                      END;
001166
001167              uClearBreaks:
001168                  IF clickPanel.view.isPrintable THEN
001169                      BEGIN
001170                      clickPanel.view.printManager.ClearPageBreaks(FALSE);
001171                      clickPanel.currentView.ReDoBreaks;
001172                      clickPanel.Invalidate;  {later, do a more selective inval}
001173                      END;
001174
001175              {$IFC fDbgABC}
001176              {Debug Menu}
001177              uReportEvents:
001178                  SELF.ToggleFlag(eventDebug);
001179              uCountHeap:
001180                  SELF.ToggleFlag(fCountHeap);
001181              uCheckIndices:
001182                  SELF.ToggleFlag(fCheckIndices);
001183              uDumpGlobals:                                       {dump process variables}
001184                  process.DumpGlobals;
001185              uDumpPrelude:                                       {dump active document's prelude}
001186                  document.DumpPrelude;
001187              uExperimenting:
001188                  SELF.ToggleFlag(fExperimenting);
001189              uReptGarbage, uFreeGarbage:
001190                  IF document <> clipboard THEN
001191                      BEGIN
001192                      MarkHeap(document.docHeap, ORD(document.dataSegment.preludePtr^.docDirectory));
001193                      SweepHeap(document.docHeap, cmdNumber = uReptGarbage);
001194                      END;
001195              uMainScramble:
```

```
001196                  THz(mainHeap)^.fScramble := NOT THz(mainHeap)^.fScramble;
001197              uDocScramble:
001198                  IF currentDocument <> NIL THEN
001199                      THz(currentDocument.docHeap)^.fScramble :=
001200                          NOT THz(currentDocument.docHeap)^.fScramble;
001201              {$ENDC}
001202
001203              OTHERWISE
001204                  BEGIN
001205                  IF menuBar.GetCmdName(cmdNumber, @str) THEN
001206                      process.ArgAlert(1, str)
001207                  ELSE
001208                      BEGIN
001209                      {$IFC fDbgABC}
001210                      ABCbreak('called new command, but no command in menu', cmdNumber);
001211                      {$ENDC}
001212                      process.ArgAlert(1, 'Unknown Command');
001213                      END;
001214
001215                  IF selection.kind = nothingKind THEN
001216                      process.Stop(phNoSel)
001217                  ELSE
001218                      process.Stop(phUnkCmd);
001219                  END;
001220              END;
001221
001222          IF permCmd THEN
001223              BEGIN
001224              command := TCommand.CREATE(NIL, SELF.Heap, cmdNumber, NIL, FALSE, revealNone);
001225              WITH command DO
001226                  BEGIN
001227                  unHiliteBefore[doPhase] := FALSE;
001228                  hiliteAfter[doPhase] := FALSE;
001229                  END;
001230              NewStdCommand := command;
001231              END;
001232
001233          IF error > 0 THEN
001234              process.Stop(process.Phrase(error));
001235          {$IFC fTrace}EP;{$ENDC}
001236      END;
001237      {$S SgABCres}
001238
001239
001240      {$S sCommand}
001241      PROCEDURE {TWindow.}PerformCommand{(newCommand: TCommand)};
001242      BEGIN
001243          {$IFC fTrace}BP(7);{$ENDC}
```

```
001244          IF newCommand <> NIL THEN {this is a command that changes the document}
001245              BEGIN
001246              {commit the previous command}
001247              SELF.CommitLast;
001248
001249              {save the new command & get rid of the old one}
001250              SELF.SaveCommand(newCommand);
001251
001252              {execute the new command}
001253              SELF.PerformLast(doPhase);
001254              END;
001255          {$IFC fTrace}EP;{$ENDC}
001256      END;
001257      {$S SgABCres}
001258
001259
001260      {$S sCommand}
001261      PROCEDURE {TWindow.}PerformLast{(cmdPhase: TCmdPhase)};  {+SW+} {LSR: Your version below, commented out}
001262      VAR image:      TImage;
001263          lastCmd:    TCommand;
001264          lastWindow: TWindow;
001265      BEGIN
001266          {$IFC fTrace}BP(7);{$ENDC}
001267          IF SELF <> currentWindow THEN
001268              currentWindow.PerformLast(cmdPhase)
001269          ELSE
001270              BEGIN
001271              lastCmd := SELF.lastCmd;
001272              image := lastCmd.image;
001273              IF image = NIL THEN
001274                  lastWindow := SELF
001275              ELSE
001276                  lastWindow := image.view.panel.window; {+SW+}
001277
001278              {UnHighlight all selections before performing the command (unless command object says otherwise)}
001279              IF lastCmd.unHiliteBefore[cmdPhase] THEN
001280                  currentWindow.selectWindow.Highlight(hOnToOff);  {+sw+}
001281
001282              IF cmdPhase <> doPhase THEN
001283                  lastWindow.RestoreSelection;{+sw+}
001284
001285              IF lastCmd.revelation <> revealNone THEN
001286                  lastWindow.RevealSelection(lastCmd.revelation = revealAll,
001287                                             NOT lastCmd.unHiliteBefore[cmdPhase]);
001288
001289              lastWindow.selectPanel.selection.PerformCommand(lastCmd, cmdPhase);  {+sw+}
001290
001291              {Save selection in each panel; hilite if necessary}
```

```
001292              SELF.SaveSelection;
001293
001294              IF NOT deferUpdate THEN
001295                  IF lastCmd.HiliteAfter[cmdPhase] THEN
001296                      BEGIN
001297                      lastWindow.Update(FALSE);{+sw+}
001298                      lastWindow.Highlight(hOffToOn);{+sw+}
001299                      END
001300                  ELSE
001301                      lastWindow.Update(TRUE);{+sw+}
001302              END;
001303          {$IFC fTrace}EP;{$ENDC}
001304      END;
001305
001306
001307  (*  PROCEDURE {TWindow.}PerformLast{(cmdPhase: TCmdPhase)};
001308      VAR lastCmd:    TCommand;
001309
001310          PROCEDURE PerformIt;
001311              BEGIN
001312              {UnHighlight all selections before performing the command (unless command object says otherwise)}
001313              IF lastCmd.unHiliteBefore[cmdPhase] THEN
001314                  SELF.Highlight(hOnToOff);
001315
001316              IF cmdPhase <> doPhase THEN
001317                  SELF.RestoreSelection;
001318
001319              IF lastCmd.revelation <> revealNone THEN
001320                  SELF.RevealSelection(lastCmd.revelation = revealAll, NOT lastCmd.unHiliteBefore[cmdPhase]);
001321
001322              lastCmd.doing := cmdPhase <> undoPhase;
001323              lastCmd.Perform(cmdPhase);
001324
001325              {Save selection in each panel; hilite if necessary}
001326              SELF.SaveSelection;
001327
001328              IF lastCmd.HiliteAfter[cmdPhase] THEN
001329                  BEGIN
001330                  SELF.Update(FALSE);
001331                  SELF.Highlight(hOffToOn);
001332                  END
001333              ELSE
001334                  SELF.Update(TRUE);
001335          END;
001336
001337      BEGIN
001338          {$IFC fTrace}BP(7);{$ENDC}
001339          IF SELF <> currentWindow THEN
```

```
001340              currentWindow.PerformLast(cmdPhase)
001341          ELSE
001342              BEGIN
001343              lastCmd := SELF.lastCmd;
001344              PerformIt;
001345              END;
001346          {$IFC fTrace}EP;{$ENDC}
001347      END;
001348  *)
001349
001350      {$S sStartup}
001351      PROCEDURE {TWindow.}PickStdCursor;
001352      BEGIN
001353          {$IFC fTrace}BP(7);{$ENDC}
001354          SetStdCursor(arrowCursor);
001355          {$IFC fTrace}EP;{$ENDC}
001356      END;
001357      {$S SgABCres}
001358
001359
001360  {$IFC LibraryVersion > 20}
001361  {$S SgABCcld}
001362      PROCEDURE {TWindow.}PrefixToName(VAR error, offset: INTEGER; VAR prefix, name: TFilePath);
001363      BEGIN
001364          {$IFC fTrace}BP(7);{$ENDC}
001365          PrefixToName(error, offset, WindowPtr(SELF.wmgrID), Pathname(prefix), Pathname(name));
001366          {$IFC fTrace}EP;{$ENDC}
001367      END;
001368  {$S SgABCres}
001369  {$ENDC}
001370
001371
001372      {$S SgABCpri}
001373      PROCEDURE {TWindow.}Print{(panel: TPanel; nixPgRange: BOOLEAN; nixWholeDialog: BOOLEAN)};
001374      VAR prPrfAlias:     TPrPrfAlias;
001375          indeedPrint:    BOOLEAN;
001376          isNewStyle:     BOOLEAN;
001377          document:       TDocManager;
001378  {$IFC libraryVersion <= 20}         { P E P S I }
001379          error:          INTEGER;
001380          prIns:          TPrIns;
001381  {$ELSEC}                            { S P R I N G }
001382          prIns:          TPrRec;
001383          prMode:         PrMenuSuppress;
001384  {$ENDC}
001385      BEGIN
001386          {$IFC fTrace}BP(7);{$ENDC}
001387          IF currentDocument <> NIL THEN
```

```
001388              document := currentDocument
001389          ELSE
001390              document := boundDocument;
001391
001392          IF document = clipboard THEN
001393              prPrfAlias.reserve := clipPrintPref
001394          ELSE
001395              prPrfAlias.reserve := document.dataSegment.preludePtr^.PrintPref;
001396
001397          PushFocus;
001398  {$IFC libraryVersion <= 20}         { P E P S I }
001399          indeedPrint :=  FPrInsDlg(error, prPrfAlias.prPrf, prPrfAlias.prIns, isNewStyle);
001400  {$ELSEC}                            { S P R I N G }
001401          IF nixWholeDialog THEN
001402              prMode := ePrDialogSuppress
001403          ELSE
001404          IF nixPgRange THEN
001405              prMode := ePgRangeSuppress
001406          ELSE
001407              prMode := ePrNormal;
001408
001409          indeedPrint := FPrInsDlg(prPrfAlias.prPrf, isNewStyle, prMode);
001410  {$ENDC}
001411          PopFocus;
001412
001413  {$IFC libraryVersion <= 20}         { P E P S I }
001414          IF error > 0 THEN
001415              process.Stop(phUnknown)     {PrMgr passed on an OS error}
001416          ELSE
001417  {$ENDC}
001418              BEGIN
001419              IF isNewStyle THEN  {style changed during print-instance dialog}
001420                  SELF.AcceptNewPrinterInfo(document, prPrfAlias.reserve);
001421
001422              IF indeedPrint THEN
001423                  BEGIN
001424                  amPrinting := TRUE;
001425                  PushFocus;
001426                  panel.PrintView(prPrfAlias.reserve);
001427                  amPrinting := FALSE;
001428                  PopFocus;
001429
001430                  SELF.Update(TRUE); {clear out white area from RECORDING box}
001431  {$IFC libraryVersion <= 20}         { P E P S I }
001432                  PrBgdDlg(error, FALSE);   {put up background dialog}
001433  {$ENDC}  {NB: For Spring, user-interface says we go back to the app, not to the background dialog}
001434                  END;
001435              END;
```

```
001436          {$IFC fTrace}EP;{$ENDC}
001437      END;
001438      {$S SgABCres}
001439
001440
001441      {$S SgABCcld}
001442      PROCEDURE {TWindow.}PutUpDialogBox{(dialogBox: TDialogBox)};
001443      BEGIN
001444          {$IFC fTrace}BP(7);{$ENDC}
001445          SELF.CommitLast;
001446          SELF.dialogBox := dialogBox;
001447  (*      IF dialogBox.selectWindow <> NIL THEN
001448              SELF.selectWindow := dialogBox.selectWindow;  *)  {+SW+}
001449          dialogBox.Appear;
001450          {$IFC fTrace}EP;{$ENDC}
001451      END;
001452      {$S SgABCres}
001453
001454
001455  {$S sStartup}
001456      PROCEDURE {TWindow.}Refresh{(rActions: TActions; highTransit: THighTransit)};
001457
001458          PROCEDURE RefreshPanel(obj: TObject);
001459              VAR panel:  TPanel;
001460          BEGIN
001461              panel := TPanel(obj);
001462              IF RectIsVisible(panel.outerRect) THEN
001463                  panel.Refresh(rActions, highTransit);
001464          END;
001465
001466      BEGIN
001467          {$IFC fTrace}BP(7);{$ENDC}
001468          IF SELF = clipboard.window THEN
001469              highTransit := hNone;
001470          {$IFC fDbgABC}
001471          IF (rBackground IN rActions) AND (highTransit > hOffToOn) THEN
001472              ABCBreak('Refresh: rBackground requested, but highTransit does not start from Off', 0);
001473          {$ENDC}
001474          IF rFrame IN rActions THEN
001475              SELF.Frame;
001476          SELF.panels.Each(RefreshPanel);
001477          {$IFC fTrace}EP;{$ENDC}
001478      END;
001479  {$S SgABCres}
001480
001481
001482      {$S sStartup}
001483      PROCEDURE {TWindow.}Resize{(moving: BOOLEAN)};
```

```
001484              {Make the Tool Kit data structures agree with the window manager's idea of the window size;
001485                  also, ensure that bottom right corner of window is on the screen}
001486          VAR oldOuterRect:       Rect;
001487              myGrafPort:         GrafPtr;
001488              newScreenRect:      Rect;
001489              proposedSize:       Point;
001490              minExtent:          Point;
001491              newOuterRect:       Rect;
001492              currentlyVisible:   BOOLEAN;
001493      BEGIN
001494          {$IFC fTrace}BP(7);{$ENDC}
001495          PushFocus;
001496          SELF.Focus;
001497
001498          currentlyVisible := SELF.IsVisible;
001499          myGrafPort:= POINTER(SELF.wmgrID);
001500
001501          IF currentlyVisible THEN
001502              BEGIN
001503              { Find out where the window is on the screen }
001504              newScreenRect := myGrafPort^.portRect;
001505
001506              proposedSize := Point(FDiagRect(newScreenRect));
001507
001508              IF NOT SELF.believeWmgr THEN
001509                  WITH SELF DO
001510                      BEGIN
001511                      maxInnerSize := proposedSize;
001512                      believeWmgr := TRUE;
001513                      END;
001514
001515              IF moving THEN { Constrain it to the maximum explicitly set by the user }
001516                  BEGIN
001517                  LocalToGlobal(newScreenRect.topLeft);
001518                  { Propose the window botRight be at the screen botRight }
001519                  proposedSize := Point(FPtMinPt(Point(FPtMinusPt(screenBits.bounds.botRight,
001520                                                                  newScreenRect.topLeft)),
001521                                                 SELF.maxInnerSize));
001522                  END;
001523              END
001524          ELSE
001525              proposedSize := Point(FDiagRect(SELF.innerRect));
001526
001527          { But be sure it is at least the minimum size }
001528          SELF.GetMinExtent(minExtent, TRUE);
001529          proposedSize := Point(FPtMaxPt(proposedSize, minExtent));
001530
001531          IF NOT moving THEN
```

```
001532              SELF.maxInnerSize := proposedSize;
001533
001534          oldOuterRect := SELF.outerRect;
001535          SetRect(newScreenRect, 0, 0, proposedSize.h, proposedSize.v);
001536
001537          IF currentlyVisible THEN
001538              { finally set the wmgr window ("folder") size. }
001539              FolderSize(myGrafPort, proposedSize.h, proposedSize.v, FALSE);
001540
001541          { Reset our idea of window's size }
001542          SELF.SetInnerRect(newScreenRect);
001543
001544          ClipRect(SELF.innerRect);
001545          focusRgn := thePort^.visRgn;
001546          newOuterRect := SELF.outerRect;
001547          IF NOT EqualPt(oldOuterRect.botRight, newOuterRect.botRight) THEN
001548              SELF.panelTree.ResizeOutside(newOuterRect);
001549
001550          PopFocus;
001551          {$IFC fTrace}EP;{$ENDC}
001552      END;
001553      {$S SgABCres}
001554
001555
001556      {$S SgABCcld}
001557      PROCEDURE {TWindow.}ResizeTo{(newSize: Point)};
001558      BEGIN
001559          {$IFC fTrace}BP(7);{$ENDC}
001560          IF NOT EqualPt(Point(FDiagRect(SELF.innerRect)), newSize) THEN
001561              BEGIN
001562              FolderSize(POINTER(SELF.wmgrID), newSize.h, newSize.v, FALSE);
001563              SELF.Resize(FALSE);
001564              END;
001565          {$IFC fTrace}EP;{$ENDC}
001566      END;
001567      {$S SgABCres}
001568
001569
001570
001571      {$S sRes}
001572      PROCEDURE {TWindow.}RestoreSelection;
001573          PROCEDURE RestoreSel(obj: TObject);
001574          BEGIN
001575              TPanel(obj).undoSelection.Restore;  {$}
001576          END;
001577
001578      BEGIN
001579          {$IFC fTrace}BP(7);{$ENDC}
```

```
001580          SELF.selectPanel := SELF.undoSelPanel;
001581          SELF.clickPanel := SELF.undoClickPanel;
001582          SELF.selectWindow := SELF.undoSelWindow; {+SW+}
001583          SELF.panels.Each(RestoreSel);
001584          IF SELF.dialogBox <> NIL THEN
001585              SELF.dialogBox.RestoreSelection;
001586          {$IFC fTrace}EP;{$ENDC}
001587      END;
001588
001589
001590
001591      {$S sCommand}
001592      PROCEDURE {TWindow.}RevealSelection(asMuchAsPossible, doHilite: BOOLEAN);
001593          PROCEDURE RevlSel(obj: TObject);
001594          BEGIN
001595              TPanel(obj).selection.Reveal(asMuchAsPossible);
001596          END;
001597
001598      BEGIN
001599          {$IFC fTrace}BP(7);{$ENDC}
001600          SELF.Update(doHilite);
001601          SELF.panels.Each(RevlSel);
001602          SELF.Update(doHilite);
001603          {$IFC fTrace}EP;{$ENDC}
001604      END;
001605
001606
001607      {$S sCommand}
001608      PROCEDURE {TWindow.}SaveCommand{(command: TCommand)};
001609
001610          PROCEDURE SaveUndoSelection(obj: TObject);
001611              VAR panel: TPanel;
001612                  sel:   TSelection;
001613          BEGIN
001614              panel := TPanel(obj);
001615              sel := panel.undoSelection.FreedAndReplacedBy(TSelection(panel.selection.Clone(SELF.Heap)));
001616          END;
001617
001618      BEGIN  {Called by PerformCommand between NewCommand & PerformLast to establish an undo-point}
001619          {$IFC fTrace}BP(7);{$ENDC}
001620          IF SELF <> currentWindow THEN
001621              currentWindow.SaveCommand(command)  {probably this is a dialog box}
001622          ELSE
001623          IF SELF.lastCmd <> NIL THEN
001624              SELF.lastCmd.Become(command)
001625          ELSE
001626              SELF.lastCmd := command;
001627
```

```
001628          SELF.panels.Each(SaveUndoSelection);
001629          {$IFC fTrace}EP;{$ENDC}
001630      END;
001631
001632
001633      {$S sCommand}
001634      PROCEDURE {TWindow.}SaveSelection;
001635          PROCEDURE SaveSel(obj: TObject);
001636          BEGIN
001637              TPanel(obj).selection.Save;
001638          END;
001639
001640      BEGIN
001641          {$IFC fTrace}BP(7);{$ENDC}
001642          SELF.panels.Each(SaveSel);
001643          SELF.undoSelPanel := SELF.selectPanel;
001644          SELF.undoClickPanel := SELF.clickPanel;
001645          SELF.undoSelWindow := SELF.selectWindow; {+SW+}
001646          IF SELF.dialogBox <> NIL THEN
001647              SELF.dialogBox.SaveSelection;
001648          {$IFC fTrace}EP;{$ENDC}
001649      END;
001650  {$}
001651
001652
001653      {$S sCommand}
001654      PROCEDURE {TWindow.}SetupMenus;
001655          VAR anS255:     S255;
001656              undoTempl:  TCmdNumber;
001657              mapHandle:  TMapHandle;
001658              selection:  TSelection;
001659              i:          INTEGER;
001660              wmgrCmd:    TWmgrCmd;
001661              checkIt:    BOOLEAN;
001662              mainWindow: TWindow;
001663      BEGIN  {NOTE: wmgrMenus[menuIndex] can not be assigned to a local variable because it is passed as a VAR}
001664          {$IFC fTrace}BP(5);{$ENDC}
001665          mainWindow := currentWindow;
001666
001667          {First, change the text of the Set Aside and Undo items.}
001668          mainWindow.GetTitle(anS255); {don't use SELF because we might be a dialog box}
001669          anS255 := CONCAT('"', anS255, '"');
001670          menuBar.BuildCmdName(uSetAside, utSetAside, @anS255);
001671          menuBar.BuildCmdName(uSetClipAside, utSetAside, @anS255);
001672
001673          IF mainWindow.lastCmd = NIL THEN {the mainWindow always has the last command}
001674              menuBar.BuildCmdName(uUndoLast, utUndoLast, NIL)
001675          ELSE
```

```
001676              BEGIN
001677              IF mainWindow.lastCmd.doing THEN
001678                  undoTempl := utUndoLast
001679              ELSE
001680                  undoTempl := utRedoLast;
001681
001682              IF menuBar.GetCmdName(mainWindow.lastCmd.cmdNumber, @anS255) THEN
001683                  BEGIN
001684                  anS255 := CONCAT('"', anS255, '"');
001685                  menuBar.BuildCmdName(uUndoLast, undoTempl, @anS255);
001686                  END
001687              ELSE
001688                  menuBar.BuildCmdName(uUndoLast, undoTempl, NIL);
001689              END;
001690
001691          {Then enable and check the appropriate items}
001692          mapHandle := TMapHandle(menuBar.mapping);
001693          selection := SELF.selectPanel.selection;
001694
001695          FOR i := 1 TO menuBar.numCommands DO
001696              BEGIN
001697              wmgrCmd := mapHandle^^.table[i];
001698              WITH wmgrCmd DO
001699                  IF menuBar.isLoaded[menuIndex] THEN
001700                      BEGIN
001701                      checkIt := FALSE;
001702
001703                      (**********
001704                      IF selection.CanDoCommand(cmdNumber, checkIt) THEN
001705                          EnableItem(wmgrMenus[menuIndex], itemIndex)
001706                      ELSE
001707                          DisableItem(wmgrMenus[menuIndex], itemIndex);
001708                      **********)
001709
001710                     {The following line is an optimization for the preceding}
001711                      wmgrMenus[menuIndex].enableFlags[itemIndex] :=
001712                          selection.CanDoCommand(cmdNumber, checkit);
001713                      CheckItem(wmgrMenus[menuIndex], itemIndex, checkIt);
001714                      END;
001715              END;
001716          {$IFC fTrace}EP;{$ENDC}
001717      END;
001718
001719
001720      {$S SgABCini}
001721      PROCEDURE {TWindow.}SetWmgrId{(itsWmgrId: TWindowID)};
001722      VAR panelScanner:   TListScanner;
001723          panel:          TPanel;
```

```
001724          paneScanner:    TListScanner;
001725          pane:           TPane;
001726      BEGIN
001727          {$IFC fTrace}BP(7);{$ENDC}
001728          SELF.wmgrId := itsWmgrId;
001729          panelScanner := SELF.panels.Scanner;
001730          WHILE panelScanner.Scan(panel) DO
001731              BEGIN
001732              paneScanner := panel.panes.Scanner;
001733              WHILE paneScanner.Scan(pane) DO
001734                  pane.port := POINTER(itsWmgrId);
001735              END;
001736          {$IFC fTrace}EP;{$ENDC}
001737      END;
001738      {$S SgABCres}
001739
001740
001741      {$S SgABCcld}
001742      PROCEDURE {TWindow.}StashPicture{(highTransit: THighTransit)};
001743      BEGIN
001744          {$IFC fTrace}BP(7);{$ENDC}
001745          RectRgn(altVisRgn, SELF.outerRect);
001746          useAltVisRgn := TRUE;   {Make TPad.Focus use altVisRgn instead of visRgn}
001747
001748          PushFocus;
001749          SELF.Focus;
001750
001751          WMOpenPicture(POINTER(SELF.wmgrID));
001752          SELF.Refresh([rErase, rFrame, rBackground, rDraw], highTransit); {recorded & not displayed}
001753          WMClosePicture;
001754
001755          useAltVisRgn := FALSE;
001756
001757          PopFocus;
001758          {$IFC fTrace}EP;{$ENDC}
001759      END;
001760      {$S SgABCres}
001761
001762
001763      {$S SgABCcld}
001764      PROCEDURE {TWindow.}TakeDownDialogBox;  {+sw+}
001765          VAR dialogBox: TDialogBox;
001766      BEGIN
001767          {$IFC fTrace}BP(7);{$ENDC}
001768          {Don't CommitLast here, because the Dialog Box may have created a command that can be undone later}
001769          dialogBox := SELF.dialogBox;
001770          IF dialogBox <> NIL THEN
001771              BEGIN
```

```
001772              IF SELF.lastCmd <> NIL THEN
001773                  IF SELF.lastCmd.image <> NIL THEN
001774                      IF SELF.lastCmd.image.view.panel.window = dialogBox THEN
001775                          SELF.CommitLast;  {+sw+}
001776
001777              dialogBox.Disappear;
001778              IF dialogBox.freeOnDismissal THEN
001779                  SELF.objectToFree := dialogBox; {+SW+}   {will be freed at end of event loop}
001780              SELF.dialogBox := NIL;
001781              SELF.selectWindow := SELF;
001782              END
001783          ELSE
001784              ABCBreak('TakeDownDialogBox, but none up', 0);
001785          {$IFC fTrace}EP;{$ENDC}
001786      END;
001787      {$S SgABCres}
001788
001789
001790      {$IFC fDbgABC}
001791      {$S SgABCdbg}
001792      PROCEDURE {TWindow.}ToggleFlag{(VAR flag: BOOLEAN)};
001793      BEGIN
001794          {$IFC fTrace}BP(1);{$ENDC}
001795          flag := NOT flag;
001796          {$IFC fTrace}EP;{$ENDC}
001797      END;
001798      {$S SgABCres}
001799      {$ENDC}
001800
001801
001802      {$S SgABCcld}
001803      PROCEDURE {TWindow.}UndoLast;
001804      VAR lastCmd:    TCommand;
001805          str:        S255;
001806          cmdPhase:   TCmdPhase;
001807      BEGIN
001808          {$IFC fTrace}BP(7);{$ENDC}
001809          IF SELF <> currentWindow THEN
001810              currentWindow.UndoLast
001811          ELSE
001812              BEGIN
001813              lastCmd := SELF.lastCmd;
001814              IF lastCmd = NIL THEN
001815                  process.Stop(phNoCommand)
001816              ELSE
001817              IF NOT lastCmd.undoable THEN
001818                  BEGIN
001819                  IF NOT menuBar.GetCmdName(lastCmd.cmdNumber, @str) THEN
```

```
001820                      BEGIN
001821                      {$IFC fDbgABC} ABCbreak('TCommand.cmdNumber not in menu', lastCmd.cmdNumber); {$ENDC}
001822                      str := 'Last Command';
001823                      END;
001824                  process.ArgAlert(1, str);
001825                  process.Stop(phCantUndo);
001826                  END
001827              ELSE
001828              IF lastCmd.doing THEN
001829                  SELF.PerformLast(undoPhase)
001830              ELSE
001831                  SELF.PerformLast(redoPhase);
001832              END;
001833          {$IFC fTrace}EP;{$ENDC}
001834      END;
001835      {$S SgABCres}
001836
001837
001838      {$S sStartup}
001839      PROCEDURE {TWindow.}Update{(doHilite: BOOLEAN)};
001840          VAR pWindow:        WindowPtr;
001841              updateRgn:      RgnHandle;
001842              highTransit:    THighTransit;
001843      BEGIN
001844          {$IFC fTrace}BP(7);{$ENDC}
001845          PushFocus;
001846          SELF.Focus;
001847          pWindow := POINTER(SELF.wmgrID);
001848          BeginUpdate(pWindow);
001849          updateRgn := pWindow^.visRgn;
001850          IF NOT EmptyRgn(updateRgn) THEN
001851              BEGIN
001852              IF doHilite THEN
001853                  highTransit := highLevel[SELF.isActive]
001854              ELSE
001855                  highTransit := hNone;
001856              FillRgn(updateRgn, white);
001857              SELF.Refresh([rFrame, rBackground, rDraw], highTransit);
001858              END;
001859          EndUpdate(pWindow);
001860          PopFocus;
001861          {$IFC fTrace}EP;{$ENDC}
001862      END;
001863  {$S SgABCres}
001864
001865
001866      {$S sStartup}
001867      FUNCTION  {TWindow.}WantMenu{(menuID: INTEGER; inClipboard: BOOLEAN): BOOLEAN};
```

```
001868      BEGIN
001869          {$IFC fTrace}BP(3);{$ENDC}
001870          IF inClipboard THEN
001871              WantMenu := menuID = mnuClipFilePrint
001872          ELSE
001873              WantMenu := (menuID < mBuzzword);
001874          {$IFC fTrace}EP;{$ENDC}
001875      END;
001876      {$S SgABCres}
001877
001878
001879  {$S SgABCini}
001880  END;
001881  {$S SgABCres}
001882
001883
001884  METHODS OF TDialogBox;
001885
001886
001887      {$S SgABCcld}
001888      FUNCTION  {TDialogBox.}CREATE{(object: TObject; heap: THeap; itsResizability: BOOLEAN; itsHeight: INTEGER;
001889                                     itsKeyResponse, itsMenuResponse,
001890                                     itsDownInMainWindowResponse: TDiResponse): TDialogBox};
001891          VAR diBxRect: Rect;
001892      BEGIN
001893          {$IFC fTrace}BP(7);{$ENDC}
001894          IF object = NIL THEN
001895              object := NewObject(heap, THISCLASS);
001896          SELF := TDialogBox(TWindow.CREATE(object, heap, ORD(dialogFolder), itsResizability));
001897
001898          WITH SELF DO
001899              BEGIN
001900              keyResponse := itsKeyResponse;
001901              menuResponse := itsMenuResponse;
001902              downInMainWindowResponse := itsDownInMainWindowResponse;
001903              freeOnDismissal := FALSE; {+SW+}
001904              END;
001905          SELF.GetPrinterMetrics; {mostly just so that these won't be total garbage in debug output}
001906          SetRect(diBxRect, 0, 0, screenBits.bounds.right, itsHeight);
001907          SELF.SetInnerRect(diBxRect);
001908          {$IFC fTrace}EP;{$ENDC}
001909      END;
001910      {$S SgABCres}
001911
001912
001913      {$IFC fDebugMethods}
001914      {$S SgABCdbg}
001915      PROCEDURE {TDialogBox.}Fields{(PROCEDURE Field(nameAndType: S255))};
```

```
001916      BEGIN
001917          TWindow.Fields(Field);
001918          Field('keyResponse: Byte');
001919          Field('menuResponse: Byte');
001920          Field('downInMainWindowResponse: Byte');
001921          Field('freeOnDismissal: BOOLEAN');  {+SW+}
001922          Field('');
001923      END;
001924      {$S SgABCres}
001925      {$ENDC}
001926
001927
001928      {$S SgABCcld}
001929      PROCEDURE {TDialogBox.}Appear;
001930      BEGIN
001931          {$IFC fTrace}BP(7);{$ENDC}
001932          DialogHeight(LengthRect(SELF.innerRect, v), TRUE);
001933          SELF.outerRect.bottom := SELF.outerRect.top; {force Resize to recalculate everything}
001934          SELF.Resize(FALSE);
001935          {$IFC fTrace}EP;{$ENDC}
001936      END;
001937
001938
001939      {$S SgABCcld}
001940      PROCEDURE {TDialogBox.}BeDismissed;
001941      BEGIN
001942          {$IFC fTrace}BP(7);{$ENDC}
001943          currentWindow.TakeDownDialogBox;
001944          {$IFC fTrace}EP;{$ENDC}
001945      END;
001946
001947
001948      {$S SgABCcld}
001949      PROCEDURE {TDialogBox.}Disappear;
001950      BEGIN
001951          {$IFC fTrace}BP(7);{$ENDC}
001952          DialogHeight(0, FALSE);
001953          SELF.believeWmgr := FALSE; {the window's innerRect is known to NOT match the size of the dialog box}
001954          {$IFC fTrace}EP;{$ENDC}
001955      END;
001956
001957
001958      {$S SgABCcld}
001959      PROCEDURE {TDialogBox.}GetMinExtent{(VAR minExtent: Point; windowIsResizingIt: BOOLEAN)};
001960      BEGIN
001961          {$IFC fTrace}BP(9);{$ENDC}
001962          SUPERSELF.GetMinExtent(minExtent, windowIsResizingIt);
001963          minExtent.h := screenBits.bounds.right;
```

```
001964          {$IFC fTrace}EP;{$ENDC}
001965      END;
001966      {$S SgABCres}
001967
001968
001969      {$S SgABCcld}
001970      FUNCTION  {TDialogBox.}IsVisible{: BOOLEAN};
001971          VAR info:   WindowInfo;
001972      BEGIN
001973          {$IFC fTrace}BP(3);{$ENDC}
001974          IF SUPERSELF.IsVisible THEN
001975              IsVisible := currentWindow.dialogBox = SELF
001976          ELSE
001977              IsVisible := FALSE;
001978          {$IFC fTrace}EP;{$ENDC}
001979      END;
001980      {$S SgABCres}
001981
001982
001983  {$S SgABCini}
001984  END;
001985  {$S SgABCres}
001986
001987
001988  {SUBROUTINES OF TMenuBar}
001989
001990
001991      {$S sRes}
001992  PROCEDURE InAllMenusDo{(iffLoaded: BOOLEAN; theCommand: TCmdNumber;
001993                          PROCEDURE doProc(VAR menu: MenuInfo; itemIndex: INTEGER))};
001994      VAR i:          INTEGER;
001995          lowIDX:     INTEGER;
001996          highIDX:    INTEGER;
001997          mapHandle:  TMapHandle;
001998          fFound:     BOOLEAN;
001999  BEGIN
002000      fFound := FALSE;
002001      mapHandle := TMapHandle(menuBar.mapping);
002002      lowIDX := 1;
002003      highIDX := menuBar.numCommands;
002004
002005      WHILE NOT fFound AND (lowIdx <= highIdx) DO
002006          BEGIN
002007          i := (lowIDX+highIDX) DIV 2;
002008    {$R-} WITH mapHandle^^.table[i] DO {$IFC fRngABC}{$R+}{$ENDC} { OK to do this because once
002009                                  we call doProc, we don't refer to this record any more }
002010              IF theCommand = cmdNumber THEN
002011                  BEGIN
```

```
002012                  fFound := TRUE;
002013                  IF menuBar.isLoaded[menuIndex] = iffLoaded THEN
002014                      doProc(wmgrMenus[menuIndex], itemIndex);
002015                  END
002016              ELSE
002017              IF theCommand > cmdNumber THEN
002018                  lowIDX := i+1
002019              ELSE
002020                  highIDX := i-1;
002021          END;
002022  END;
002023
002024
002025      {$S sCommand}
002026  FUNCTION  CmdFromWmgr(menuId, itemIndex: INTEGER): TCmdNumber;
002027      VAR wmgrCmd:    TWmgrCmd;
002028          cmdNumber:  TCmdNumber;
002029          i:          INTEGER;
002030          mapHandle:  TMapHandle;
002031  BEGIN {does not need to be very fast}
002032          {$IFC fMaxTrace}BP(1);{$ENDC}
002033          {$IFC fMaxTrace}EP;{$ENDC}
002034      IF itemIndex < 0 THEN
002035          CmdFromWmgr := -itemIndex {this is how we will implement graphical menus}
002036      ELSE
002037          BEGIN
002038          mapHandle := TMapHandle(menuBar.mapping);
002039          FOR i := 1 TO menuBar.numCommands DO
002040              BEGIN
002041              {$R-}
002042              wmgrCmd := mapHandle^^.table[i];
002043              {$IFC fRngABC}{$R+}{$ENDC}
002044              IF wmgrCmd.itemIndex = itemIndex THEN
002045                  IF menuBar.isLoaded[wmgrCmd.menuIndex] THEN
002046                      IF wmgrMenus[wmgrCmd.menuIndex].menuId = menuId THEN
002047                          BEGIN
002048                          CmdFromWmgr := wmgrCmd.cmdNumber;
002049                          EXIT(CmdFromWmgr);
002050                          END;
002051               END;
002052          CmdFromWmgr := 0;
002053          END;
002054  END;
002055
002056
002057      {$S sRes}
002058  FUNCTION  FindMenu(menuID: INTEGER): INTEGER;
002059      { given a menuID (the number in the phrase file) return the menuIndex into
```

```
002060          our array of menuInfo records }
002061  VAR menuIndex:  INTEGER;
002062  BEGIN
002063          {$IFC fMaxTrace}BP(1);{$ENDC}
002064          {$IFC fMaxTrace}EP;{$ENDC}
002065      FOR menuIndex := 1 TO menuBar.numMenus DO
002066          IF wmgrMenus[menuIndex].menuID = menuID THEN
002067              BEGIN
002068              FindMenu := menuIndex;
002069              EXIT(FindMenu);
002070              END;
002071      FindMenu := 0;
002072  END;
002073
002074
002075  METHODS OF TMenuBar;
002076
002077
002078      {$S SgABCini}
002079      FUNCTION  {TMenuBar.}CREATE{(object: TObject; heap: THeap; itsScanner: TFileScanner): TMenuBar};
002080          VAR menu:           MenuInfo;
002081              numMenus:       INTEGER;
002082              i:              INTEGER;
002083              numBytes:       INTEGER;
002084              mapping:        TArray;
002085              numCommands:    INTEGER;
002086      BEGIN
002087          {$IFC fTrace}BP(7);{$ENDC}
002088          IF object = NIL THEN
002089              object := NewObject(heap, THISCLASS);
002090          SELF := TMenuBar(object);
002091
002092          menu.drawProc := @drawTxtMenu;
002093          menu.chooseProc := @chooseTxtItem;
002094          numMenus := itsScanner.ReadNumber(2);
002095          SELF.numMenus := numMenus;
002096          FOR i := 1 TO numMenus DO
002097              BEGIN
002098              menu.menuId := itsScanner.ReadNumber(2);
002099              itsScanner.XferSequential(xRead, @menu.enableFlags, 4);
002100              numBytes := itsScanner.ReadNumber(2);
002101              menu.menuData := POINTER(ORD(HAllocate(POINTER(ORD(heap)), numBytes)));
002102              itsScanner.XferSequential(xRead, @menu.menuData^^, numBytes);
002103              CalcMenuSize(menu);
002104              wmgrMenus[i] := menu;
002105              SELF.isLoaded[i] := FALSE;
002106              END;
002107          mapping := POINTER(ORD(itsScanner.ReadArray(heap, SIZEOF(TWmgrCmd))));
```

```
002108          SELF.mapping := mapping;
002109          numCommands := mapping.Size;
002110          SELF.numCommands := numCommands;
002111          InitErrorAbort(itsScanner.error);
002112          {$IFC fTrace}EP;{$ENDC}
002113      END;
002114      {$S SgABCres}
002115
002116
002117      {$IFC fDebugMethods}
002118      {$S SgABCdbg}
002119      PROCEDURE {TMenuBar.}Fields{(PROCEDURE Field(nameAndType: S255))};
002120      BEGIN
002121          Field('isLoaded: ARRAY [1..31] OF BOOLEAN');   (* MaxMenus = 31 *)
002122          Field('mapping: TArray');
002123          Field('numMenus: INTEGER');
002124          Field('numCommands: INTEGER');
002125      END;
002126      {$S SgABCres}
002127      {$ENDC}
002128
002129
002130      {$S sRes}
002131      PROCEDURE {TMenuBar.}BuildCmdName{(destCmd, templateCmd: TCmdNumber; param: TPString)};
002132      VAR templ:  S255;
002133          xStart: INTEGER;
002134          xEnd:   INTEGER;
002135      BEGIN
002136          {$IFC fTrace}BP(3);{$ENDC}
002137          IF SELF.GetCmdName(templateCmd, @templ) THEN
002138              BEGIN
002139              xStart := POS('^', templ);
002140              IF xStart > 0 THEN
002141                  BEGIN
002142                  DELETE(templ, xStart, 1);
002143
002144                  xEnd := POS('^', templ);
002145                  IF xEnd > 0 THEN
002146                      DELETE(templ, xEnd,1)
002147                  ELSE
002148                      xEnd := LENGTH(templ) + 1;
002149
002150                  IF param <> NIL THEN
002151                      BEGIN
002152                      DELETE(templ, xStart, xEnd-xStart);
002153                      INSERT(param^, templ, xStart);
002154                      END;
002155                  END;
```

```
002156
002157              SELF.PutCmdName(destCmd, @templ);
002158              END;
002159          {$IFC fTrace}EP;{$ENDC}
002160      END;
002161
002162
002163      {$S sRes}
002164      PROCEDURE {TMenuBar.}Check{(cmdNumber: TCmdNumber; checked: BOOLEAN)};
002165        Label 1;
002166          PROCEDURE DoCheck(VAR menu: MenuInfo; itemIndex: INTEGER);
002167          BEGIN
002168              CheckItem(menu, itemIndex, checked);
002169              Goto 1;
002170          END;
002171      BEGIN
002172          {$IFC fTrace}BP(2);{$ENDC}
002173          InAllMenusDo(TRUE, cmdNumber, DoCheck);
002174      1:  {$IFC fTrace}EP;{$ENDC}
002175      END;
002176
002177
002178      {$S sCommand}
002179      FUNCTION  {TMenuBar.}CmdKey{(ch: CHAR): TCmdNumber};
002180          VAR menuId, itemIndex: INTEGER;
002181      BEGIN
002182          {$IFC fTrace}BP(7);{$ENDC}
002183          MenuKey(ch, menuId, itemIndex);
002184          if menuId <> 0 THEN
002185              HiLiteMenu(menuId);
002186          CmdKey := CmdFromWmgr(menuId, itemIndex);
002187          {$IFC fTrace}EP;{$ENDC}
002188      END;
002189
002190
002191      {$S sRes}
002192      PROCEDURE {TMenuBar.}Delete{(menuID: INTEGER)};
002193          VAR menuIndex:  INTEGER;
002194      BEGIN
002195          {$IFC fTrace}BP(7);{$ENDC}
002196          DeleteMenu(menuId);
002197          menuIndex := FindMenu(menuID);
002198          IF menuIndex > 0 THEN
002199              SELF.isLoaded[menuIndex] := FALSE;
002200          {$IFC fTrace}EP;{$ENDC}
002201      END;
002202
002203
```

```
002204      {$S sCommand}
002205      FUNCTION  {TMenuBar.}DownAt{(mousePt: Point): TCmdNumber};
002206          VAR menuId, itemIndex: INTEGER;
002207      BEGIN
002208          {$IFC fTrace}BP(7);{$ENDC}
002209          process.ChangeCursor(arrowCursor);
002210          MenuSelect(mousePt, menuId, itemIndex);
002211          if menuId <> 0 THEN
002212              HiLiteMenu(menuId);
002213          DownAt := CmdFromWmgr(menuId, itemIndex);
002214          {$IFC fTrace}EP;{$ENDC}
002215      END;
002216
002217
002218      {$S sStartup}
002219      PROCEDURE {TMenuBar.}Draw;
002220      BEGIN
002221          {$IFC fTrace}BP(7);{$ENDC}
002222          DrawMenuBar;
002223          {$IFC fTrace}EP;{$ENDC}
002224      END;
002225
002226
002227      {$S sRes}
002228      PROCEDURE {TMenuBar.}Enable{(cmdNumber: TCmdNumber; canBeChosen: BOOLEAN)};
002229      BEGIN
002230          {$IFC fTrace}BP(2);{$ENDC}
002231          IF canBeChosen THEN
002232              InAllMenusDo(TRUE, cmdNumber, EnableItem)
002233          ELSE
002234              InAllMenusDo(TRUE, cmdNumber, DisableItem);
002235          {$IFC fTrace}EP;{$ENDC}
002236      END;
002237
002238
002239      {$S sRes}
002240      PROCEDURE {TMenuBar.}EndCmd;
002241      BEGIN
002242          {$IFC fTrace}BP(6);{$ENDC}
002243          HiLiteMenu(0);
002244          {$IFC fTrace}EP;{$ENDC}
002245      END;
002246
002247
002248      {$S sRes}
002249      FUNCTION  {TMenuBar.}GetCmdName{(cmdNumber: TCmdNumber; pName: TPString): BOOLEAN};
002250        Label 1;
002251          PROCEDURE DoGet(VAR menu: MenuInfo; itemIndex: INTEGER);
```

```
002252              VAR kludge: Str255;
002253          BEGIN
002254              IF pName <> NIL THEN
002255                  BEGIN
002256                  GetItem(menu, itemIndex, @kludge);
002257                  XferLeft(@kludge, POINTER(ORD(pName)), LENGTH(kludge)+1);
002258                  END;
002259              Goto 1;
002260          END;
002261      BEGIN
002262          {$IFC fTrace}BP(6);{$ENDC}
002263          GetCmdName := TRUE;
002264          InAllMenusDo(TRUE, cmdNumber, DoGet);
002265          InAllMenusDo(FALSE, cmdNumber, DoGet);
002266          GetCmdName := FALSE;
002267          IF pName <> NIL THEN
002268              pName^ := '';
002269      1:  {$IFC fTrace}EP;{$ENDC}
002270      END;
002271
002272
002273      {$S sRes}
002274      PROCEDURE {TMenuBar.}HighlightMenu(withCmd: TCmdNumber);
002275          LABEL   1;
002276
002277          PROCEDURE DoHighlight(VAR menu: MenuInfo; itemIndex: INTEGER);
002278          BEGIN
002279              HiLiteMenu(menu.menuID);
002280              Goto 1;
002281          END;
002282      BEGIN
002283          {$IFC fTrace}BP(6);{$ENDC}
002284          InAllMenusDo(TRUE, withCmd, DoHighlight);
002285  1:      {$IFC fTrace}EP;{$ENDC}
002286      END;
002287
002288
002289      {$S sStartup}
002290      PROCEDURE {TMenuBar.}Insert{(menuID, beforeId: INTEGER)};
002291          VAR menuIndex:  INTEGER;
002292      BEGIN
002293          {$IFC fTrace}BP(7);{$ENDC}
002294          menuIndex := FindMenu(menuID);
002295          IF menuIndex > 0 THEN
002296              BEGIN
002297              InsertMenu(wmgrMenus[menuIndex], beforeId);
002298              SELF.isLoaded[menuIndex] := TRUE;
002299              END;
```

```
002300          {$IFC fTrace}EP;{$ENDC}
002301      END;
002302
002303
002304      {$S SgABCcld}
002305      FUNCTION  {TMenuBar.}MenuWithID(menuID: INTEGER): Ptr;
002306          VAR menuIndex:  INTEGER;
002307      BEGIN
002308          {$IFC fTrace}BP(7);{$ENDC}
002309          menuIndex := FindMenu(menuID);
002310          IF menuIndex > 0 THEN
002311              MenuWithId := @wmgrMenus[menuIndex]
002312          ELSE
002313              MenuWithID := NIL;
002314          {$IFC fTrace}EP;{$ENDC}
002315      END;
002316
002317
002318      {$S sRes}
002319      PROCEDURE {TMenuBar.}PutCmdName{(cmdNumber: TCmdNumber; pName: TPString)};
002320          Label 1;
002321          VAR kludge: Str255;
002322          PROCEDURE DoPut(VAR menu: MenuInfo; itemIndex: INTEGER);
002323          BEGIN
002324              SetItem(menu, itemIndex, @kludge);
002325              Goto 1;
002326          END;
002327      BEGIN
002328          {$IFC fTrace}BP(6);{$ENDC}
002329          XferLeft(POINTER(ORD(pName)), @kludge, LENGTH(pName^)+1);
002330          InAllMenusDo(TRUE, cmdNumber, DoPut);
002331          InAllMenusDo(FALSE, cmdNumber, DoPut);
002332      1:  {$IFC fTrace}EP;{$ENDC}
002333      END;
002334
002335
002336  (**********
002337  {$S SgABCini}
002338      PROCEDURE {TMenuBar.}SetupGrMenu(menuID: INTEGER; width, height: INTEGER;
002339                                       newChooseProc, newDrawProc: Ptr);
002340          {if either proc is NIL, don't change the current value;
002341           if either width or height is <= 0, don't change the current value;
002342           when the menu is first read in, it is setup to behave like a standard text menu}
002343          VAR menuIndex:  INTEGER
002344      BEGIN
002345          {$IFC fTrace}BP(7);{$ENDC}
002346          menuIndex := FindMenu(menuID);
002347          IF menuIndex > 0 THEN
```

```
002348              WITH wmgrMenus[menuIndex] DO
002349                  BEGIN
002350                  IF width > 0 THEN
002351                      menuWidth := width;
002352                  IF height > 0 THEN
002353                      menuHeight := height;
002354                  IF newChooseProc <> NIL THEN
002355                      chooseProc := newChooseProc;
002356                  IF newDrawProc <> NIL THEN
002357                      drawProc := newDrawProc;
002358                  END;
002359          {$IFC fTrace}EP;{$ENDC}
002360      END;
002361  {$S SgABCres}
002362  **********)
002363
002364
002365      {$S sRes}
002366      PROCEDURE {TMenuBar.}Unload;
002367          VAR i:  INTEGER;
002368      BEGIN
002369          {$IFC fTrace}BP(7);{$ENDC}
002370          ClearMenuBar;
002371          FOR i := 1 TO SELF.numMenus DO
002372              SELF.isLoaded[i] := FALSE;
002373          {$IFC fTrace}EP;{$ENDC}
002374      END;
002375
002376
002377  {$S SgABCini}
002378  END;
002379  {$S SgABCres}
002380
002381
002382  {$IFC LibraryVersion <= 20 AND FALSE} {do it this way in case we need it back for the Pepsi version}
002383  METHODS OF TFont;
002384
002385
002386      {$S SgABCini}
002387      FUNCTION  {TFont.}CREATE{(object: TObject; heap: THeap; itsFamily: INTEGER): TFont};
002388      BEGIN
002389          {$IFC fTrace}BP(7);{$ENDC}
002390          IF object = NIL THEN
002391              object := NewObject(heap, THISCLASS);
002392          SELF := TFont(object);
002393
002394          SELF.family := itsFamily;
002395          {$IFC fTrace}EP;{$ENDC}
```

```
002396      END;
002397      {$S SgABCres}
002398
002399
002400      {$IFC fDebugMethods}
002401      {$S SgABCdbg}
002402      PROCEDURE {TFont.}Fields{(PROCEDURE Field(nameAndType: S255))};
002403      BEGIN
002404          Field('family: INTEGER');
002405      END;
002406      {$S SgABCres}
002407      {$ENDC}
002408
002409
002410  {$S SgABCini}
002411  END;
002412  {$S SgABCres}
002413  {$ENDC}
002414
002415
002416
002417

End of File -- Lines: 2417 Characters: 73342
```

```
================================================================================================
FILE: "LIBTK/UABC5.TEXT"
================================================================================================

000001  {INCLUDE FILE UABC5 -- IMPLEMENTATION OF UABC}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004                      {TPanel-TBand-TPane-TMarginPad-TBodyPad-TScroller-TScrollBar}
000005
000006  {changed 05/11/84 11:25 In TPanel.MoveSplitBefore, if we are creating a new split check the new band's
000007                           ViewLCd after creation; if not the same as what we passed in, invalidate the
000008                           new band's innerRect.}
000009
000010
000011
000012  {$S sCldInit}
000013  PROCEDURE InvalDiffRect(r1, r2: Rect); {invalidate r1 - r2}
000014      VAR dummyRect:  Rect;
000015          rgn1:       RgnHandle;
000016          rgn2:       RgnHandle;
000017
000018      BEGIN
000019          {$IFC fTrace}BP(5);{$ENDC}
000020          IF EmptyRect(r1) THEN
000021              {nothing to do}
000022          ELSE IF SectRect(r1, r2, dummyRect) THEN
000023              BEGIN
000024              rgn1 := NewRgn;
000025              rgn2 := NewRgn;
000026              RectRgn(rgn1, r1);
000027              RectRgn(rgn2, r2);
000028              DiffRgn(rgn1, rgn2, rgn1);
000029              InvalRgn(rgn1);
000030              DisposeRgn(rgn1);
000031              DisposeRgn(rgn2);
000032              END
000033          ELSE
000034              InvalRect(r1);
000035
000036          {$IFC fTrace}EP;{$ENDC}
000037      END;
000038
000039
000040  METHODS OF TPanel;
000041
000042
000043      {$S SgABCini}
```

```
000044      FUNCTION  {TPanel.}CREATE{(object: TObject; heap: THeap; itsWindow: TWindow;
000045                                  minHeight, minWidth: INTEGER; itsVAbilities, itsHAbilities: TAbilities)
000046                                  : TPanel};
000047          VAR hasWinResize:   BOOLEAN;
000048              viewedLRect:    LRect;
000049              panes:          TList;
000050              bandOuterRect:  Rect;
000051              vhs:            VHSelect;
000052              scrollBar:      TScrollBar;
000053              scroller:       TScroller;
000054              band:           TBand;
000055              bandList:       TList {OF TBand};
000056              aPane:          TPane;
000057      BEGIN
000058          {$IFC fTrace}BP(7);{$ENDC}
000059          IF object = NIL THEN
000060              object := NewObject(heap, THISCLASS);
000061          SELF := TPanel(object);
000062
000063          IF aSplit IN itsVAbilities THEN
000064              itsVAbilities := itsVAbilities + [aBar];
000065          IF aSplit IN itsHAbilities THEN
000066              itsHAbilities := itsHAbilities + [aBar];
000067
000068          WITH SELF DO
000069              BEGIN
000070              window := itsWindow;
000071              view := NIL;
000072              currentView := NIL;
000073              selection := NIL;
000074              undoSelection := NIL;
000075              minInnerDiagonal.v := minHeight;
000076              minInnerDiagonal.h := minWidth;
000077              abilities[v] := itsVAbilities;
000078              abilities[h] := itsHAbilities;
000079              previewMode := mPrvwOff;
000080              paginatedView := NIL;
000081              parentBranch := NIL;
000082              resizeBranch := NIL;
000083              scrollBars[v] := NIL; {so GetBorder (called by SetOuterRect below) won't blow up}
000084              scrollBars[h] := NIL; {ditto}
000085
000086              {$H-}
000087              SetPt(tlSideBandSize, -1, -1); {+++ LSR +++}
000088              SetPt(brSideBandSize, -1, -1); {+++ LSR +++}
000089              {$H+}
000090
000091              deletedSplits := NIL;
```

```
000092              END;
000093
000094            SELF.zoomed := FALSE;
000095            WITH SELF.zoomFactor DO
000096          {$H-} BEGIN
000097                SetPt(numerator, 1, 1);
000098                SetPt(denominator, 1, 1);
000099          {$H+} END;
000100
000101            SELF.SetOuterRect(itsWindow.outerRect);
000102
000103            noPad.RectToLRect(SELF.innerRect, viewedLRect);
000104            panes := TList.CREATE(NIL, heap, 1);
000105            SELF.panes := panes;
000106            aPane := SELF.NewPane(heap, SELF.innerRect, viewedLRect);
000107            WITH SELF.lastClick DO
000108                BEGIN
000109                gotPane := TRUE;
000110                clickPane := aPane;
000111                END;
000112            SELF.panes.InsLast(aPane);
000113
000114            bandOuterRect := SELF.innerRect;
000115            InsetRect(bandOuterRect, -1, -1);
000116
000117            FOR vhs := v TO h DO
000118                BEGIN
000119                scrollBar := TScrollBar.CREATE(NIL, heap, vhs, bandOuterRect, aBar IN SELF.abilities[vhs]);
000120                SELF.scrollBars[vhs] := scrollBar;
000121                scroller := scrollBar.firstBox;
000122                band := SELF.NewBand(heap, SELF.innerRect, scroller, vhs);
000123                band.panes.InsLast(SELF.panes.First);
000124                bandList := TList.CREATE(NIL, heap, 1);
000125                bandList.InsLast(band);
000126                SELF.bands[vhs] := bandList;
000127                END;
000128
000129            IF itsWindow.panelTree = NIL THEN {The first panel gets inserted automatically}
000130                BEGIN
000131                itsWindow.panelTree := SELF;
000132                itsWindow.panels.InsLast(SELF);
000133
000134                itsWindow.selectPanel := SELF;
000135                itsWindow.clickPanel := SELF;
000136
000137                SELF.DecideAboutBars(SELF.outerRect);
000138                END;
000139            {$IFC fTrace}EP;{$ENDC}
```

```
000140      END;
000141      {$S SgABCres}
000142
000143
000144      {$S SgABCini}
000145      PROCEDURE {TPanel.}Free;
000146          VAR vhs: VHSelect;
000147      BEGIN
000148          {$IFC fTrace}BP(7);{$ENDC}
000149          Free(SELF.selection);
000150          Free(SELF.undoSelection);
000151          Free(SELF.view);
000152          IF SELF.currentView <> SELF.view THEN
000153              Free(SELF.currentView);
000154          FOR vhs := v TO h DO
000155              BEGIN
000156              SELF.bands[vhs].Free;
000157              SELF.scrollBars[vhs].Free;
000158              END;
000159          SELF.panes.Free;
000160          Free(SELF.deletedSplits);
000161          TArea.Free;
000162          {$IFC fTrace}EP;{$ENDC}
000163      END;
000164      {$S SgABCres}
000165
000166
000167      {$IFC fDebugMethods}
000168      {$S SgABCdbg}
000169      PROCEDURE {TPanel.}Fields{(PROCEDURE Field(nameAndType: S255))};
000170      BEGIN
000171          TArea.Fields(Field);
000172          Field('window: TWindow');
000173          Field('panes: TList');
000174          Field('currentView: TView');
000175          Field('view: TView');
000176          Field('paginatedView: TPaginatedView');
000177          Field('selection: TSelection');
000178          Field('undoSelection: TSelection');
000179          Field('bands: ARRAY [0..1] OF TList');
000180          Field('scrollBars: ARRAY [0..1] OF TScrollBar');
000181          Field('abilities: ARRAY [0..1] OF Byte');
000182          Field('minInnerDiagonal: Point');
000183          Field('resizeBranch: TBranchArea');
000184          Field('zoomed: BOOLEAN');
000185          Field('zoomFactor: RECORD numerator: Point; denominator: Point END');
000186          Field('previewMode: Byte');
000187          Field('');
```

```
000188          Field('lastClick: RECORD gotPane: BOOLEAN; clickPane: TPane; END');
000189          Field('contentRect: Rect');
000190          Field('tlSideBandSize: Point');
000191          Field('brSideBandSize: Point');
000192          Field('deletedSplits: TArray');
000193          Field('');
000194      END;
000195      {$S SgABCres}
000196      {$ENDC}
000197
000198
000199      {$S sRes}
000200      PROCEDURE {TPanel.}AutoScroll{(mousePt: Point)};
000201          VAR vhs:        VHSelect;
000202              mouseCd:    INTEGER;
000203              f:          INTEGER;
000204              deltaLPt:   LPoint;
000205              pane:       TPane;
000206              r:          Rect;
000207      BEGIN
000208          {$IFC fTrace}BP(7);{$ENDC}
000209          SELF.currentView.GetStdScroll(deltaLPt);
000210          FOR vhs := v TO h DO
000211              BEGIN
000212              mouseCd := mousePt.vh[vhs];
000213              IF NOT (aScroll IN SELF.abilities[vhs]) THEN
000214                  f := 0
000215              ELSE
000216              IF mouseCd < SELF.contentRect.topLeft.vh[vhs] THEN {+++ LSR +++}
000217                  f := -1
000218              ELSE
000219              IF mouseCd > SELF.contentRect.botRight.vh[vhs] THEN {+++ LSR +++}
000220                  f := 1
000221              ELSE
000222                  f := 0;
000223              deltaLPt.vh[vhs] := f * deltaLPt.vh[vhs];
000224              END;
000225
000226          {Find the pane to scroll; make sure it is not in a side band}
000227          r := SELF.contentRect;
000228          InsetRect(r, 1, 1); {Because the outerRects of a side band's panes overlap the contentRect by 1 pixel;
000229                               (ChildWithPt checks the outerRect)}
000230          RectHavePt(r, mousePt);
000231          pane := TPane(SELF.ChildWithPt(mousePt, SELF.panes, mousePt));
000232          pane.ScrollBy(deltaLPt);
000233          {$IFC fTrace}EP;{$ENDC}
000234      END;
000235
```

```
000236
000237      {$S sRes}
000238      PROCEDURE {TPanel.}BeginSelection;   {+SW+}
000239          VAR thisWindow:         TWindow;
000240              companionWindow:    TWindow;
000241          PROCEDURE DeselPanel(obj: TObject);
000242          BEGIN
000243              TPanel(obj).selection.Deselect;
000244          END;
000245
000246      BEGIN
000247          {$IFC fTrace}BP(7);{$ENDC}
000248          SELF.BeSelectPanel(TRUE);
000249          thisWindow := SELF.window;
000250          companionWindow := NIL;
000251          IF thisWindow = currentWindow THEN
000252              companionWindow := thisWindow.dialogBox {+SW+}
000253          ELSE
000254          IF thisWindow = currentWindow.dialogBox THEN
000255              IF currentWindow.dialogBox.downInMainWindowResponse = diGiveToMainWindow THEN
000256                  companionWindow := currentWindow;
000257
000258          IF companionWindow <> NIL THEN
000259              BEGIN
000260              PushFocus;
000261              companionWindow.Focus;
000262              companionWindow.panels.Each(DeselPanel);
000263              PopFocus;
000264              END;
000265          thisWindow.panels.Each(DeselPanel);
000266          {$IFC fTrace}EP;{$ENDC}
000267      END;
000268
000269
000270      {$S sRes}
000271      PROCEDURE {TPanel.}BeSelectPanel{(inSelectWindow: BOOLEAN)};
000272      BEGIN
000273          {$IFC fTrace}BP(7);{$ENDC}
000274          IF inSelectWindow THEN
000275              currentWindow.selectWindow := SELF.window;
000276          SELF.window.selectPanel := SELF;
000277          {$IFC fTrace}EP;{$ENDC}
000278      END;
000279
000280
000281      {$S sStartup}
000282      PROCEDURE {TPanel.}CleanUpPanes{(deleteList: TList)};
000283          VAR s:      TListScanner;
```

```
000284              pane:   TPane;
000285              bs:     TListScanner;
000286              band:   TBand;
000287              vhs:    VHSelect;
000288          BEGIN
000289              {$IFC fTrace}BP(7);{$ENDC}
000290              IF SELF.lastClick.gotPane THEN
000291                  IF deleteList.Pos(0, SELF.lastClick.clickPane) > 0 THEN
000292                      WITH SELF.lastClick DO
000293                          BEGIN
000294                          gotPane := FALSE;
000295                          clickPt := clickPane.innerRect.topLeft;  {+}
000296                          END;
000297
000298              s := deleteList.Scanner;
000299              WHILE s.Scan(pane) DO
000300                  BEGIN
000301                  SELF.panes.DelObject(pane, FALSE);
000302                  FOR vhs := v To h DO
000303                      BEGIN
000304                      bs := SELF.bands[vhs].Scanner;
000305                      WHILE bs.Scan(band) DO
000306                          band.panes.DelObject(pane, FALSE);
000307                      END;
000308                  END;
000309              deleteList.Free;
000310              {$IFC fTrace}EP;{$ENDC}
000311          END;
000312          {$S SgABCres}
000313
000314
000315
000316          {$S sRes}
000317          PROCEDURE {TPanel.}ComputeContentRect;
000318          BEGIN
000319              {$IFC fTrace}BP(7);{$ENDC}
000320              WITH SELF DO
000321                  BEGIN
000322                  {$H-}
000323                  contentRect.topLeft := Point(FPtPlusPt(innerRect.topLeft, tlSideBandSize));
000324                  contentRect.botRight := Point(FPtMinusPt(innerRect.botRight, brSideBandSize));
000325
000326                  InsetRect(contentRect, 1, 1);
000327                  {$H+}
000328                  END;
000329              {$IFC fTrace}EP;{$ENDC}
000330          END;
000331
```

```
000332
000333      {$S sStartup}
000334      FUNCTION  {TPanel.}CursorAt{(mousePt: Point): TCursorNumber};
000335          VAR pane:       TPane;
000336              nearestPt:  Point;
000337      BEGIN
000338          {$IFC fTrace}BP(2);{$ENDC}
000339          IF NOT RectHasPt(SELF.outerRect, mousePt) THEN
000340              CursorAt := noCursor
000341          ELSE
000342          IF currentDocument = clipboard THEN
000343              CursorAt := arrowCursor
000344          ELSE
000345          IF RectHasPt(SELF.innerRect, mousePt) THEN
000346              BEGIN
000347              pane := TPane(SELF.ChildWithPt(mousePt, SELF.panes, nearestPt));
000348              CursorAt := pane.CursorAt(mousePt);
000349              END
000350          ELSE
000351              CursorAt := arrowCursor;
000352      {$IFC fTrace}EP;{$ENDC}
000353      END;
000354
000355
000356      {$S SgABCini}
000357      PROCEDURE {TPanel.}DecideAboutBars{(newOuterRect: Rect)};
000358          VAR branch:         TBranchArea;
000359              needsBothBars:  BOOLEAN;
000360              vhs:            VHSelect;
000361      BEGIN
000362          {$IFC fTrace}BP(7);{$ENDC}
000363          branch := SELF.FindBranchThatIsResized;
000364          SELF.resizeBranch := branch;
000365          needsBothBars := (branch <> NIL) OR
000366              (EqualPt(newOuterRect.botRight, SELF.window.outerRect.botRight) AND SELF.window.isResizable);
000367          FOR vhs := v TO h DO
000368              SELF.scrollBars[vhs].ChangeVisibility(needsBothBars, zeroRect, SELF.abilities[vhs]);
000369          SELF.SetOuterRect(newOuterRect);
000370          {$IFC fTrace}EP;{$ENDC}
000371      END;
000372      {$S SgABCres}
000373
000374
000375      {$S SgABCcld}
000376      FUNCTION  {TPanel.}Divide{(vhs: VHSelect;
000377                                 fromEdgeOfPanel: INTEGER; units: TUnitsFromEdge;
000378                                 whoCanResizeIt: TResizability;
000379                                 minSize: INTEGER; itsVAbilities, itsHAbilities: TAbilities): TPanel};
```

```
000380
000381          VAR itsMinInnerDiag:    Point;
000382              panel:              TPanel;     {the new panel}
000383
000384      BEGIN
000385          {$IFC fTrace}BP(7);{$ENDC}
000386          itsMinInnerDiag := SELF.minInnerDiagonal;
000387          itsMinInnerDiag.vh[vhs] := minSize;
000388
000389          panel := TPanel.CREATE(NIL, SELF.heap, SELF.window,
000390                                 itsMinInnerDiag.v, itsMinInnerDiag.h, itsVAbilities, itsHAbilities);
000391
000392          SELF.Insert(panel, vhs, fromEdgeOfPanel, units, whoCanResizeIt);
000393
000394          Divide := panel;
000395          {$IFC fTrace}EP;{$ENDC}
000396      END;
000397      {$S SgABCres}
000398
000399
000400      {$S sScroll}
000401      PROCEDURE {TPanel.}DoScrolling{(inArea: TArea; itsPane: TPane;
000402                              hOk, vOk: BOOLEAN; VAR deltaLPt: LPoint)};
000403              {positive scrolls towards end, (0,0) means invalidate only;
000404                  if inArea is a pane then itsPane=inArea
000405                  if inArea is a band then itsPane is any one of the band's panes;
000406                  hOk & vOk indicate whether scrolling is allowed in that direction;
000407                  deltaLPt is set to amount actually scrolled by;
000408
000409              NOTE: assumes we are focused on something at least as big as inArea. }
000410          VAR viewedLRect:        LRect;
000411              resizing:           BOOLEAN;
000412              scrollableLRect:    LRect;
000413              freedomLRect:       LRect;
000414              deltaPt:            Point;
000415              vhs:                VHSelect;
000416      BEGIN
000417          {$IFC fTrace}BP(6);{$ENDC}
000418          resizing := EqualLPt(deltaLPt, zeroLPt);
000419
000420          itsPane.GetScrollLimits(viewedLRect, scrollableLRect);
000421
000422          LRectMinusLRect(scrollableLRect, viewedLRect, freedomLRect);
000423          LRectHaveLPt(freedomLRect, deltaLPt);
000424
000425          IF NOT hOk THEN
000426              deltaLPt.h := 0;
000427          IF NOT vOK THEN
```

```
000428              deltaLPt.v := 0;
000429
000430          IF NOT EqualLPt(deltaLPt, zeroLPt) THEN
000431              BEGIN
000432              IF resizing OR NOT IsSmallPt(deltaLPt) THEN
000433                  InvalRect(inArea.innerRect)
000434              ELSE
000435                  BEGIN
000436                  itsPane.LDistToDist(deltaLPt, deltaPt);
000437                  ScrollRect(inArea.innerRect, -deltaPt.h, -deltaPt.v, scrollRgn);
000438                  InvalRgn(scrollRgn);
000439                  END;
000440             END;
000441          {$IFC fTrace}EP;{$ENDC}
000442      END;
000443
000444
000445      {$S sRes}
000446      FUNCTION  {TPanel.}DownAt{(mousePt: Point): BOOLEAN};
000447          VAR found:          BOOLEAN;
000448              cantDown:       BOOLEAN;
000449              vhs:            VHSelect;
000450              outerRect:      Rect;
000451              innerRect:      Rect;
000452              insetContent:   Rect;
000453              growRect:       Rect;
000454              window:         TWindow;
000455              dialogBox:      TDialogBox;
000456              icon:           TEnumIcons;
000457              scroller:       TScroller;
000458              pane:           TPane;
000459              viewedLRect:    LRect;
000460              hysteresis:     BOOLEAN;
000461              limitRect:      Rect;
000462              hysterPt:       Point;
000463              origPt:         Point;
000464              diffPt:         Point;
000465              nearestPt:      Point;
000466              aheadEvent:     EventRecord;
000467              destPanel:      TPanel;
000468              destView:       TView;
000469              lPtInView:      LPoint;
000470              received:       BOOLEAN;
000471              mouseInContent: BOOLEAN; {TRUE iff mouse is currently in contentRect}
000472
000473          PROCEDURE EnforceHysteresis;
000474              BEGIN
000475              diffPt := Point(FPtMinusPt(mousePt, origPt));
```

```
000476              SELF.selection.GetHysteresis(hysterPt);
000477              IF (ABS(diffPt.h) < hysterPt.h) AND (ABS(diffPt.v) < hysterPt.v) THEN
000478                  mousePt := origPt
000479              ELSE
000480                  hysteresis := FALSE;
000481              END;
000482
000483      BEGIN
000484          {$IFC fTrace}BP(7);{$ENDC}
000485          outerRect := SELF.outerRect;
000486          innerRect := SELF.innerRect;
000487          insetContent := SELF.contentRect;
000488          InsetRect(insetContent, 1, 1);
000489
000490          IF NOT RectHasPt(innerRect, mousePt) THEN  {+}
000491              BEGIN
000492              found := FALSE;
000493              FOR vhs := v TO h DO
000494                  IF NOT found THEN
000495                      IF SELF.scrollBars[vhs].DownAt(mousePt, scroller, icon) THEN
000496                          BEGIN
000497                          SELF.HitScroller(vhs, mousePt, scroller, icon);
000498                          found := TRUE;
000499                          END;
000500              IF NOT found THEN
000501                  BEGIN
000502                  SetRect(growRect, innerRect.right + 1, innerRect.bottom + 1,
000503                                    outerRect.right - 1, outerRect.bottom - 1);
000504                  IF RectHasPt(growRect, mousePt) THEN
000505                      BEGIN
000506                      SELF.DownInSizeBox(mousePt);
000507                      found := TRUE;
000508                      process.RememberCommand(uResizePanel);
000509                      END;
000510                  END;
000511
000512              DownAt := found;
000513              END
000514          ELSE
000515              BEGIN
000516              DownAt := TRUE;
000517              IF currentDocument = clipboard THEN
000518                  process.Stop(phEditClip)
000519              ELSE
000520                  BEGIN
000521                  window := SELF.window;
000522                  dialogBox := window.dialogBox;
000523                  IF dialogBox = NIL THEN
```

```
000524                  cantDown := FALSE
000525              ELSE
000526              IF dialogBox.downInMainWindowResponse = diDismissDialogBox THEN
000527                  BEGIN
000528                  dialogBox.BeDismissed;
000529                  cantDown := FALSE;
000530                  END
000531              ELSE
000532                  cantDown := (dialogBox.downInMainWindowResponse = diRefuse);
000533
000534              IF cantDown THEN
000535                  process.Stop(phDialogUp)
000536
000537              ELSE
000538
000539                  BEGIN
000540                  {$IFC fDbgABC}
000541                  IF SELF.currentView = NIL THEN
000542                      ABCBreak('DownAt with no view set', 0);
000543                  {$ENDC}
000544
000545                  mouseInContent := RectHasPt(insetContent, mousePt);
000546
000547                  pane := TPane(SELF.ChildWithPt(mousePt, SELF.panes, nearestPt));
000548                  IF mouseInContent THEN
000549                      WITH SELF.lastClick DO
000550                          BEGIN
000551                          gotPane := TRUE;
000552                          clickPane := pane;
000553                          END;
000554
000555                  process.RememberCommand(uMousePress);
000556                  pane.MouseTrack(mPress, mousePt);
000557
000558                  IF SELF.selection.canCrossPanels THEN
000559                      BEGIN
000560                      pane.RectToLRect(window.innerRect, viewedLRect);
000561
000562                      WITH pane.origin DO {$H-} {convert to (0,0)-origined view coordinates}
000563                          OffsetLRect(viewedLRect, h, v); {$H+}
000564
000565                      pane := TPane.CREATE(NIL, SELF.Heap, SELF, window.innerRect, viewedLRect);
000566                      PushFocus;
000567                      pane.Focus;
000568                      SELF.selection.DrawGhost;
000569                      PopFocus;
000570                      END;
000571
```

```
000572                      {Set up some temporaries for the StillDown loop}
000573                      limitRect := SELF.contentRect;   {AutoScroll slop}
000574                      InsetRect(limitRect, -9, -6);  {*** should be more lenient at edges of screen ***}
000575                      origPt := mousePt;
000576                      hysteresis := TRUE;
000577
000578                      WHILE stillDown do
000579                          BEGIN
000580                          GetMouse(mousePt);
000581                          {use pane.outerRect in line below, because ChildWithPt checks the outerRect}
000582                          IF NOT (RectHasPt(pane.outerRect, mousePt) OR SELF.selection.canCrossPanels) THEN
000583                              BEGIN
000584                              IF mouseInContent THEN {autoscrolling allowed}
000585                                  BEGIN
000586                                  IF NOT RectHasPt(limitRect, mousePt) THEN
000587                                      BEGIN
000588                                      SELF.AutoScroll(mousePt);
000589                                      window.Update(TRUE);
000590                                      END;
000591                                  RectHavePt(insetContent, mousePt); {force mouse point into contentRect}
000592                                  END;
000593
000594  (**** Do we want this line? Depend on if you want to allow people to down in side band, move into content
000595          and go back to side band
000596                                  mouseInContent := RectHasPt(insetContent, mousePt);
000597  ****)
000598                                  pane := TPane(SELF.ChildWithPt(mousePt, SELF.panes, mousePt));
000599                                  hysteresis := FALSE;
000600                                  END
000601                              ELSE
000602                              IF hysteresis THEN
000603                                  EnforceHysteresis;
000604                              pane.MouseTrack(mMove, mousePt);
000605                              END;
000606
000607                      IF PeekEvent(aheadEvent) THEN
000608                          BEGIN
000609                          IF aheadEvent.what = buttonUp THEN
000610                              BEGIN
000611                              mousePt := aheadEvent.where; {otherwise, use last polled point}
000612
000613                              {check to see if we've crossed the pane boundary
000614                               use pane.outerRect in line below, because ChildWithPt checks the outerRect}
000615                              IF NOT (RectHasPt(pane.outerRect, mousePt) OR
000616                                      SELF.selection.canCrossPanels) THEN
000617                                  BEGIN
000618                                  IF mouseInContent THEN  {force mouse point into contentRect}
000619                                      RectHavePt(insetContent, mousePt);
```

```
000620
000621                                  pane := TPane(SELF.ChildWithPt(mousePt, SELF.panes, mousePt));
000622                                  hysteresis := FALSE;
000623                                  END;
000624                              RectHavePt(pane.innerRect, mousePt);
000625                              END;
000626
000627                          IF hysteresis THEN
000628                              EnforceHysteresis;
000629                          END;
000630
000631                      pane.MouseTrack(mRelease, mousePt);
000632
000633                      IF SELF.selection.canCrossPanels THEN
000634                          BEGIN
000635                          pane.Free;
000636                          destPanel := TPanel(window.ChildWithPt(mousePt, window.panels, nearestPt));
000637                          IF PtInRect(mousePt, destPanel.innerRect) THEN
000638                              BEGIN
000639                              destView := destPanel.view;
000640                              pane := TPane(destPanel.ChildWithPt(mousePt, destPanel.panes, nearestPt));
000641
000642                              {Account for origin difference between window and pane}
000643                              PushFocus;
000644                              LocalToGlobal(mousePt);
000645                              pane.Focus;
000646                              GlobalToLocal(mousePt);
000647                              pane.PtToLpt(mousePt, lPtInView);
000648                              received := destView.DoReceive(SELF.selection, lPtInView);
000649                              PopFocus;
000650                              END
000651                          ELSE
000652                              received := FALSE;
000653                          IF NOT received THEN
000654                              SELF.selection.MoveBackToAnchor;
000655                          END;
000656                      END;
000657                  END;
000658              END;
000659      {$IFC fTrace}EP;{$ENDC}
000660      END;
000661  {$S SgABCres}
000662
000663
000664      {$S sRes}
000665      PROCEDURE {TPanel.}DownInSizeBox{(mousePt: Point)};
000666          VAR branch:         TBranchArea;
000667              outerRect:      Rect;
```

```
000668              oldTopLeft:     Point;
000669              oldBotRight:    Point;
000670              vhs:            VHSelect;
000671              minPt:          Point;
000672              maxPt:          Point;
000673              elderFirst:     BOOLEAN;
000674              minExtents:     ARRAY [FALSE..TRUE] OF Point;
000675              newBotRight:    Point;
000676              newCd:          INTEGER;
000677      BEGIN
000678          {$IFC fTrace}BP(7);{$ENDC}
000679          branch := SELF.resizeBranch;
000680          IF branch <> NIL THEN
000681              BEGIN
000682              outerRect := branch.outerRect;
000683              oldTopLeft := outerRect.topLeft;
000684              oldBotRight := outerRect.botRight;
000685              vhs := branch.arrangement;
000686
000687              {don't resize in the orthogonal direction}
000688              minPt := oldBotRight;
000689              maxPt := oldBotRight;
000690
000691              {limit resizing in the free direction}
000692              elderFirst := branch.elderFirst;
000693              branch.elderChild.GetMinExtent(minExtents[elderFirst], FALSE);
000694              branch.youngerChild.GetMinExtent(minExtents[NOT elderFirst], FALSE);
000695              minPt.vh[vhs] := oldTopLeft.vh[vhs]  + minExtents[TRUE].vh[vhs];
000696              maxPt.vh[vhs] := oldBotRight.vh[vhs] - minExtents[FALSE].vh[vhs];
000697
000698              {let the user specify the new botRight}
000699              ResizeFeedback(mousePt, minPt, maxPt, branch.TopLeftChild.outerRect,
000700                                  0, dhSBox, dvSBox, newBotRight);
000701
000702              newCd := newBotRight.vh[vhs];
000703              IF newCd <> oldBotRight.vh[vhs] THEN
000704                  branch.Redivide(newCd);
000705              END;
000706          {$IFC fTrace}EP;{$ENDC}
000707      END;
000708
000709
000710      {$S sRes}
000711      FUNCTION  {TPanel.}FindBranchThatIsResized{: TBranchArea};
000712          VAR child:      TArea;
000713              fini:       BOOLEAN;
000714              parent:     TBranchArea;
000715      BEGIN
```

```
000716          {$IFC fTrace}BP(7);{$ENDC}
000717          { Find the panel branch of which this is the bottom right corner of the top left child }
000718          child := SELF;
000719          fini := FALSE;
000720          REPEAT
000721              parent := child.parentBranch;
000722              IF parent = NIL THEN
000723                  fini := TRUE
000724              ELSE
000725                  fini := parent.TopLeftChild = child;
000726              child := parent;
000727            UNTIL fini;
000728
000729          FindBranchThatIsResized := NIL;
000730          IF parent <> NIL THEN
000731              IF userCanResizeIt IN parent.resizability THEN
000732                  FindBranchThatIsResized := parent;
000733          {$IFC fTrace}EP;{$ENDC}
000734      END;
000735
000736
000737      {$S sStartup}
000738      PROCEDURE {TPanel.}Frame;
000739          VAR actively:   BOOLEAN;
000740              growRect:   Rect;
000741              branch:     TBranchArea;
000742              vhs:        VHSelect;
000743              {$IFC LibraryVersion > 20}
000744              icon:       Char;
000745              {$ENDC}
000746      BEGIN
000747          {$IFC fTrace}BP(6);{$ENDC}
000748          IF NOT RectsNest(SELF.innerRect, focusRgn^^.rgnBBox) THEN
000749              BEGIN
000750              TArea.Frame;
000751              actively := SELF.window.IsActive;
000752
000753              IF SELF.scrollBars[v].isVisible OR SELF.scrollBars[h].isVisible THEN
000754                  IF NOT EqualPt(SELF.outerRect.botRight, SELF.window.outerRect.botRight) THEN
000755                      BEGIN   {Draw the panel's resize box}
000756                      SetRect(growRect, SELF.innerRect.right, SELF.innerRect.bottom,
000757                                        SELF.outerRect.right, SELF.outerRect.bottom);
000758                      FillRect(growRect, white);
000759                      IF actively THEN
000760                          BEGIN
000761                          branch := SELF.resizeBranch;
000762                          IF branch <> NIL THEN   {Draw a resize icon in the box}
000763                              BEGIN
```

```
000764                          {$IFC LibraryVersion <= 20}
000765                          vhs := branch.arrangement;
000766                          InsetRect(growRect, 3, 2);
000767                          growRect.botRight.vh[vhs] := growRect.topLeft.vh[vhs] + 1;
000768                          PenNormal;
000769                          FrameRect(growRect);
000770                          {$ELSEC}
000771                          TextFont(wmFont);
000772                          TextFace([]);
000773                          MoveTo(growRect.left, growRect.top);
000774                          IF branch.arrangement = v THEN
000775                              icon := CHR(33)
000776                          ELSE
000777                              icon := CHR(34);
000778                          DrawChar(icon);
000779                          {$ENDC}
000780                          END;
000781                      END;
000782                  END;
000783
000784              FOR vhs := v TO h DO
000785                  IF actively THEN
000786                      SELF.scrollBars[vhs].Draw
000787                  ELSE
000788                      SELF.scrollBars[vhs].Erase;
000789              END;
000790          {$IFC fTrace}EP;{$ENDC}
000791      END;
000792  {$S SgABCres}
000793
000794
000795      {$S sRes}
000796      PROCEDURE {TPanel.}GetBorder{(VAR border: Rect)};
000797          VAR vhs:    VHSelect;
000798              hasBar: BOOLEAN;
000799              d:      ARRAY[VHSelect] OF INTEGER;
000800      BEGIN
000801          {$IFC fTrace}BP(3);{$ENDC}
000802          FOR vhs := v TO h DO
000803              BEGIN
000804              IF SELF.scrollBars[vhs] = NIL THEN
000805                  hasBar := FALSE
000806              ELSE
000807                  hasBar := SELF.scrollBars[orthogonal[vhs]].isVisible;
000808
000809              IF hasBar THEN
000810                  d[vhs] := dptSbox.vh[vhs]
000811              ELSE
```

```
000812              IF SELF.outerRect.botRight.vh[vhs] = SELF.window.outerRect.botRight.vh[vhs] THEN
000813                  d[vhs] := 1
000814              ELSE
000815                  d[vhs] := 0;
000816              END;
000817          SetRect(border, -1, -1, d[h], d[v]);
000818          {$IFC fTrace}EP;{$ENDC}
000819      END;
000820
000821
000822      {$S sStartup}
000823      PROCEDURE {TPanel.}GetMinExtent{(VAR minExtent: Point; windowIsResizingIt: BOOLEAN)};
000824          VAR borderRect:     Rect;
000825      BEGIN
000826          {$IFC fTrace}BP(9);{$ENDC}
000827          RectMinusRect(SELF.outerRect, SELF.contentRect, borderRect);
000828          minExtent := Point(FPtPlusPt(SELF.minInnerDiagonal, Point(FDiagRect(borderRect))));
000829          {$IFC fTrace}EP;{$ENDC}
000830      END;
000831      {$S SgABCres}
000832
000833
000834      {$S sCldInit}
000835      PROCEDURE {TPanel.}HaveView{(view: TView)};
000836          VAR s:          TListScanner;
000837              pane:       TPane;
000838              selection:  TSelection;
000839              saveMode:   TPreviewMode;
000840      BEGIN
000841          {$IFC fTrace}BP(7);{$ENDC}
000842          saveMode := SELF.previewMode;
000843          SELF.previewMode := mPrvwOff;
000844          SELF.view := view;
000845          SELF.currentView := view;
000846
000847          s := SELF.panes.Scanner;
000848          WHILE s.Scan(pane) DO
000849              pane.HaveView(view);
000850
000851          view.BeInPanel(SELF);
000852
000853          IF SELF.selection = NIL THEN
000854              BEGIN
000855              selection := view.NoSelection;
000856              SELF.selection := selection;
000857              END
000858          ELSE
000859              SELF.selection.HaveView(view);
```

```
000860
000861          IF SELF.undoSelection = NIL THEN
000862              BEGIN
000863              selection := view.NoSelection;
000864              SELF.undoSelection := selection;
000865              END
000866          ELSE
000867              SELF.undoSelection.HaveView(view);
000868
000869          SELF.Preview(saveMode);
000870
000871          SELF.ResizeInside(SELF.innerRect); {mainly needed to force panes of a new one-panel window to size}
000872
000873          IF view.isPrintable THEN
000874              IF SELF.window.panelToPrint = NIL THEN
000875                  SELF.window.panelToPrint := SELF;
000876
000877          {$IFC fTrace}EP;{$ENDC}
000878      END;
000879      {$S SgABCres}
000880
000881
000882      {$S sRes}
000883      PROCEDURE {TPanel.}Highlight{(selection: TSelection; highTransit: THighTransit)};
000884
000885          PROCEDURE HiliteOnThePad;
000886          BEGIN
000887              selection.Highlight(highTransit);
000888          END;
000889
000890      BEGIN
000891          {$IFC fMaxTrace}BP(1);{$ENDC}
000892          {$IFC fMaxTrace}EP;{$ENDC}
000893          SELF.OnAllPadsDo(HiliteOnThePad);
000894      END;
000895  {$S SgABCres}
000896
000897
000898      {$S sScroll}
000899      PROCEDURE {TPanel.}HitScroller{(vhs: VHSelect; mousePt: Point; scroller: TScroller; icon: TEnumIcons)};
000900          VAR oldThumbPos:    INTEGER;
000901              newThumbPos:    INTEGER;
000902              deltaLStd:      LPoint;
000903              band:           TBand;
000904              newSkwrCd:      INTEGER;
000905              aScroller:      TScroller;
000906              prevScroller:   TScroller;
000907      BEGIN
```

```
000908          {$IFC fTrace}BP(7);{$ENDC}
000909          band := scroller.band;
000910          CASE icon OF
000911              iSkewer:
000912                  BEGIN
000913                  scroller.TrackSkewer(mousePt, newSkwrCd, aScroller, prevScroller);
000914                  SELF.MoveSplitBefore(scroller, newSkwrCd);
000915                  process.RememberCommand(uSplitting);
000916                  END;
000917              iThumb:
000918                  BEGIN
000919                  scroller.TrackThumb(mousePt, oldThumbPos, newThumbPos);
000920                  IF oldThumbPos <> newThumbPos THEN
000921                      BEGIN
000922                      band.ThumbTo(newThumbPos);
000923                      scroller.MoveThumb(band.ThumbPos);
000924                      END;
000925                  process.RememberCommand(uThumbing);
000926                  END;
000927              iScrollBack, iScrollFwd, iFlipBack, iFlipFwd:
000928                  BEGIN
000929                  scroller.FillIcon(icon, TRUE);
000930                  SELF.currentView.GetStdScroll(deltaLStd);
000931                  SetupMvThumb(POINTER(scroller.sBoxID));
000932                  REPEAT
000933                      band.ScrollStep(icon, deltaLStd.vh[vhs]);
000934                      SELF.window.Update(TRUE);
000935                      PenNormal;
000936                      MoveThumb(band.ThumbPos);
000937                  UNTIL NOT StillDown;
000938                  scroller.FillIcon(icon, FALSE);
000939                  process.RememberCommand(uScrolling);
000940                  END;
000941              END;
000942          {$IFC fTrace}EP;{$ENDC}
000943      END;
000944
000945
000946      {$S SgABCcld}
000947      PROCEDURE {TPanel.}Insert{(panel: TPanel; vhs: VHSelect;
000948                                 fromEdgeOfPanel: INTEGER; units: TUnitsFromEdge;
000949                                 whoCanResizeIt: TResizability)};
000950
000951          VAR window:             TWindow;
000952              elderFirst:         BOOLEAN;    {TRUE if fromEdgeOfPanel<0 (new panel below or to right of old)}
000953              myOuterRect:        Rect;       {SELF.outerRect}
000954              mySize:             INTEGER;    {Length of SELF beforehand}
000955              itsOuterRect:       Rect;       {will be panel.outerRect}
```

```
000956              newSize:            INTEGER;    {Proposed length of the new panel (in the vh direction)}
000957              cdInWindow:         INTEGER;    {the coordinate of myOuterRect that is changing}
000958              myFormerParent:     TBranchArea;
000959              ourNewParent:       TBranchArea;
000960
000961      BEGIN
000962          {$IFC fTrace}BP(7);{$ENDC}
000963          window := SELF.window;
000964          window.panels.InsLast(panel);
000965          panel.window := window;
000966
000967          elderFirst := fromEdgeOfPanel < 0;
000968          myOuterRect := SELF.outerRect;
000969          mySize := LengthRect(myOuterRect, vhs);
000970          itsOuterRect := myOuterRect;
000971
000972          newSize := ABS(fromEdgeOfPanel);
000973
000974          IF units = percentFromEdge THEN   {convert to pixelsFromEdge}
000975              newSize := LIntDivInt(LIntMulInt(mySize, newSize), 100);
000976
000977          newSize := Max(1, Min(newSize, myOuterRect.botRight.vh[vhs] - myOuterRect.topLeft.vh[vhs] - 1));
000978
000979          IF elderFirst THEN
000980              newSize := -newSize;
000981
000982          cdInWindow := TRectCoords(myOuterRect)[elderFirst].vh[vhs] + newSize;
000983          TRectCoords(myOuterRect)[elderFirst].vh[vhs] := cdInWindow;
000984          TRectCoords(itsOuterRect)[NOT elderFirst].vh[vhs] := cdInWindow;
000985
000986          myFormerParent := SELF.parentBranch;
000987
000988          ourNewParent := TBranchArea.CREATE(NIL, SELF.Heap, vhs, elderFirst, whoCanResizeIt, SELF, panel);
000989
000990          IF myFormerParent = NIL THEN
000991              window.panelTree := ourNewParent
000992          ELSE
000993              myFormerParent.ReplaceChild(SELF, ourNewParent);
000994
000995          panel.SetOuterRect(zeroRect); {since the panel is not on the screen right now,
000996                                         it shouldn't have any size}
000997          panel.ResizeOutside(itsOuterRect);
000998          SELF.ResizeOutside(myOuterRect);
000999
001000          {Just in case some panel is below its mimimum size, let the window expand if needed}
001001          window.Resize(FALSE);
001002          {$IFC fTrace}EP;{$ENDC}
001003      END;
```

```
001004      {$S SgABCres}
001005
001006
001007      {$S SgDRWres}
001008      PROCEDURE {TPanel.}Invalidate;
001009      BEGIN
001010          {$IFC fTrace}BP(7);{$ENDC}
001011          PushFocus;
001012          SELF.window.Focus;
001013          InvalRect(SELF.innerRect);
001014          PopFocus;
001015          {$IFC fTrace}EP;{$ENDC}
001016      END;
001017
001018
001019      {$S SgABCres}
001020      PROCEDURE {TPanel.}InvalLRect{(lRectInView: LRect)};
001021
001022          PROCEDURE InvalOnThePad;
001023          BEGIN
001024              thePad.InvalLRect(lRectInView);
001025          END;
001026
001027      BEGIN
001028          {$IFC fTrace}BP(7);{$ENDC}
001029          SELF.OnAllPadsDo(InvalOnThePad);
001030          {$IFC fTrace}EP;{$ENDC}
001031      END;
001032
001033
001034      {$S SgABCcld}
001035      PROCEDURE {TPanel.}MakeBand{(vhs: VHSelect; scroller, prevScroller: TScroller)};
001036          VAR prevBand:   TBand;
001037              band:       TBand;
001038      BEGIN
001039          {$IFC fTrace}BP(7);{$ENDC}
001040          prevBand := prevScroller.band;
001041          band := SELF.NewBand(SELF.Heap, zeroRect, scroller, vhs);
001042          band.panes.Become(prevBand.panes.Clone(SELF.Heap));
001043          SELF.bands[vhs].InsAt(SELF.bands[vhs].Pos(0, prevBand) + 1, band);
001044          {$IFC fTrace}EP;{$ENDC}
001045      END;
001046      {$S SgABCres}
001047
001048
001049      {$S sSplit}
001050      PROCEDURE {TPanel.}MoveSplitBefore{(scroller: TScroller; newSkwrCd: INTEGER)};
001051          VAR vhs:                        VHSelect;
```

```
001052              outsideContent:             BOOLEAN;
001053              hsb:                        THsb;
001054              prevHsb:                    THsb;
001055              prevScroller:               TScroller;
001056              nextScroller:               TScroller;
001057              otherBand:                  TBand;
001058              band:                       TBand;
001059              oldSkwrCd:                  INTEGER;
001060              newViewLCd:                 LONGINT;
001061              viewDeltaLCd:               LONGINT;   {-gb}
001062              sbRect:                     Rect;
001063              newSkwrPt:                  Point;
001064              sbList:                     TSbList;
001065              limitRect:                  Rect;
001066              r:                          Rect;
001067
001068          PROCEDURE InvalScrollers(firstBand, lastBand: TBand);
001069              VAR firstSbRect:    Rect;
001070                  lastSbRect:     Rect;
001071          BEGIN
001072              firstBand.scroller.GetSize(firstSbRect);
001073              lastBand.scroller.GetSize(lastSbRect);
001074              UnionRect(firstSbRect, lastSbRect, firstSbRect);
001075              InvalRect(firstSbRect);
001076          END;
001077
001078      BEGIN
001079          {$IFC fTrace}BP(7);{$ENDC}
001080          vhs := scroller.ScrollDir;
001081
001082          outsideContent := TRUE;
001083          WITH SELF.contentRect DO
001084              IF newSkwrCd <= topLeft.vh[vhs] THEN
001085                  newSkwrCd := topLeft.vh[vhs] - 1
001086              ELSE IF newSkwrCd >= botRight.vh[vhs] THEN
001087                  newSkwrCd := botRight.vh[vhs] + 1
001088              ELSE
001089                  outsideContent := FALSE;
001090
001091          hsb := Pointer(scroller.sBoxID);
001092          prevHsb := HsbPrev(hsb);
001093
001094          IF prevHsb = hsbNil THEN
001095              BEGIN
001096              prevScroller := NIL;
001097
001098              {make scroller refer to the scroller we are going to split}
001099              scroller.GetSize(sbRect);
```

```
001100              newSkwrPt.vh[vhs] := newSkwrCd;
001101              newSkwrPt.vh[orthogonal[vhs]] := sbRect.topLeft.vh[orthogonal[vhs]];
001102
001103              PreSbList(sbList, scroller.scrollBar);
001104              hsb := HsbFromPt(sbList, newSkwrPt);
001105              PostSbList(sbList, scroller.scrollBar);
001106
001107              IF (hsb = hsbNil)  {user started to create a new split but changed his mind} OR
001108                 outsideContent {new split would be in a side band} THEN
001109                  scroller := NIL {user started to create a new split but changed his mind}
001110              ELSE
001111                  scroller := TScroller(RefconSb(hsb));
001112              END
001113          ELSE
001114              BEGIN
001115              prevScroller := TScroller(RefconSb(prevHsb));
001116
001117              {don't allow the new position of split to cross another split}
001118              FixRLimits(hsb, limitRect);
001119              WITH limitRect DO
001120                  newSkwrCd := Max(topLeft.vh[vhs], Min(botRight.vh[vhs], newSkwrCd));
001121              END;
001122
001123
001124          IF scroller <> NIL THEN
001125              BEGIN
001126              scroller.GetSize(sbRect);
001127              oldSkwrCd := sbRect.topLeft.vh[vhs];
001128
001129              WITH SELF.contentRect DO
001130                  IF oldSkwrCd <= topLeft.vh[vhs] THEN
001131                      oldSkwrCd := topLeft.vh[vhs] - 1
001132                  ELSE IF oldSkwrCd >= botRight.vh[vhs] THEN
001133                      oldSkwrCd := botRight.vh[vhs] + 1;
001134
001135              IF newSkwrCd <> oldSkwrCd THEN
001136                  BEGIN
001137                  band := scroller.band;
001138                  viewDeltaLCd := newSkwrCd - oldSkwrCd;                                  {-gb}
001139                  IF SELF.zoomed THEN     { if zoomed then adjust viewDeltaLCd accordingly }  {-gb}
001140                      WITH SELF.zoomFactor DO                                             {-gb}
001141             {$H-}        viewDeltaCd := LIntOvrInt(LIntMulInt(viewDeltaLCd, denominator.vh[vhs]),
001142                                                    {-gb +++LSR+++} numerator.vh[vhs]);
001143             {$H-}
001144                  newViewLCd := band.ViewLCd + viewDeltaLCd;
001145                  IF prevScroller = NIL THEN
001146                      BEGIN {new band}
001147                      IF hsb <> hsbNil THEN
```

```
001148                          BEGIN
001149                          InvalScrollers(band, band);
001150
001151                          scroller.SplitAt(newSkwrCd, nextScroller);
001152
001153                          SELF.ResizeBand(vhs, band, band.ViewLCd, FALSE);
001154                          SELF.MakeBand(vhs, nextScroller, scroller);
001155                          otherBand := nextScroller.band;
001156                          SELF.ResizeBand(vhs, otherBand, newViewLCd, FALSE);
001157
001158                          {must invalidate now (special case)}
001159                          IF otherBand.ViewLCd <> newViewLCd THEN {the new band scrolled a bit}
001160                              InvalRect(otherBand.innerRect);
001161
001162                          Pt2Rect(band.innerRect.botRight, otherBand.innerRect.topLeft, r);
001163                          InvalRect(r);
001164
001165                          SELF.RepaneOrthogonalBands(vhs);
001166                          SELF.RemakePanes;
001167                          END;
001168                      END
001169                  ELSE
001170                      BEGIN {resize or delete band}
001171                      {If new position of split is outside the contentRect, make it go away}
001172                      IF outsideContent THEN
001173                          WITH limitRect DO
001174                              IF newSkwrCd <= SELF.contentRect.topLeft.vh[vhs] THEN
001175                                  newSkwrCd := topLeft.vh[vhs]
001176                              ELSE
001177                                  newSkwrCd := botRight.vh[vhs];
001178
001179                      scroller.ResplitAt(newSkwrCd, prevScroller);
001180                      otherBand := prevScroller.band;
001181                      InvalScrollers(otherBand, band);
001182
001183                      SELF.ResizeBand(vhs, otherBand, otherBand.ViewLCd, TRUE);
001184                      SELF.ResizeBand(vhs, band, newViewLCd, TRUE);
001185                      END;
001186                  END;
001187              END;
001188          {$IFC fTrace}EP;{$ENDC}
001189      END;
001190
001191
001192      {$S sCldInit}
001193      FUNCTION  {TPanel.}NewBand{(heap: THeap; myInnerRect: Rect;
001194                                  scroller: TScroller; vhs: VHSelect): TBand};
001195      BEGIN
```

```
001196          {$IFC fTrace}BP(7);{$ENDC}
001197          NewBand := TBand.CREATE(NIL, heap, SELF, myInnerRect, scroller, vhs);
001198          {$IFC fTrace}EP;{$ENDC}
001199      END;
001200      {$S SgABCres}
001201
001202
001203      {$S SgABCcld}
001204      FUNCTION {TPanel.}NewStatusView{(object: TObject; itsExtent: LRect): TView};
001205      BEGIN
001206          {$IFC fTrace}BP(7);{$ENDC}
001207          NewStatusView := TView.CREATE(object, SELF.Heap,  SELF, itsExtent, NIL, zeroLRect,
001208                                        FALSE, screenRes, TRUE);
001209          {$IFC fTrace}EP;{$ENDC}
001210      END;
001211      {$S SgABCres}
001212
001213
001214      {$S sCldInit}
001215      FUNCTION {TPanel.}NewView{(object: TObject; itsExtent: LRect; itsPrintManager: TPrintManager;
001216                                 itsDfltMargins: LRect; itsFitPerfectlyOnPages: BOOLEAN): TView};
001217      BEGIN
001218          {$IFC fTrace}BP(7);{$ENDC}
001219          NewView := TView.CREATE(object, SELF.Heap,  SELF, itsExtent, itsPrintManager, itsDfltMargins,
001220                          itsFitPerfectlyOnPages, screenRes, TRUE);
001221          {$IFC fTrace}EP;{$ENDC}
001222      END;
001223      {$S SgABCres}
001224
001225
001226      {$S sStartup}
001227      FUNCTION  {TPanel.}NewPane{(heap: THeap; innerRect: Rect; viewedLRect: LRect): TPane};
001228      BEGIN
001229          {$IFC fTrace}BP(7);{$ENDC}
001230          NewPane := TPane.CREATE(NIL, heap, SELF, innerRect, viewedLRect);
001231          {$IFC fTrace}EP;{$ENDC}
001232      END;
001233      {$S SgABCres}
001234
001235
001236      {$S SgABCcld}
001237      FUNCTION  {TPanel.}OKToDrawIn{(lRectInView: LRect): BOOLEAN};
001238      BEGIN
001239          {$IFC fTrace}BP(6);{$ENDC}
001240          IF NOT SELF.view.OKToDrawIn(lRectInView) THEN
001241              OKToDrawIn := FALSE
001242          ELSE
001243          IF SELF.previewMode = mPrvwBreaks THEN
```

```
001244              OKToDrawIn := FALSE {This will be smarter some day}
001245          ELSE
001246              OKToDrawIn := TRUE;
001247          {$IFC fTrace}EP;{$ENDC}
001248      END;
001249
001250
001251  {$S SgABCres}
001252      PROCEDURE {TPanel.}OnAllPadsDo{(PROCEDURE DoOnThePad)};
001253          VAR panes:  TList {OF TPane};
001254              pane:   TPane;
001255
001256          PROCEDURE YouDo(obj: TObject);
001257          BEGIN
001258              TPad(obj).Focus;
001259              DoOnThePad;
001260          END;
001261
001262          PROCEDURE YouDoOnPages(obj: TObject);
001263          BEGIN
001264              TPane(obj).Focus;
001265              SELF.paginatedView.DoOnPages(NOT SELF.paginatedView.workingInMargins, DoOnThePad);
001266              {i.e., if we're operating in the margins, do NOT focus on the interior}
001267          END;
001268
001269      BEGIN
001270          {$IFC fTrace}BP(7);{$ENDC}
001271          panes := SELF.panes;
001272          pane := TPane(panes.First);
001273          PushFocus;
001274
001275          IF (panes.Size = 1) AND (SELF.previewMode <> mPrvwMargins) THEN
001276              BEGIN
001277              pane.Focus;
001278              DoOnThePad;
001279              END
001280          ELSE
001281              BEGIN
001282              IF SELF.previewMode = mPrvwMargins THEN
001283                  SELF.panes.Each(YouDoOnPages)
001284              ELSE
001285                  SELF.panes.Each(YouDo);
001286              END;
001287
001288          PopFocus;
001289          {$IFC fTrace}EP;{$ENDC}
001290      END;
001291  {$S SgABCres}
```

```
001292
001293
001294      {$S sRes}
001295      FUNCTION  {TPanel.}PaneShowing{(anLRect: LRect): TPane};
001296          VAR pane:               TPane;
001297              s:                  TListScanner;
001298              viewedLRect:        LRect;
001299              scrollableLRect:    LRect;
001300      BEGIN
001301          {$IFC fTrace}BP(7);{$ENDC}
001302          PaneShowing := NIL;
001303
001304          s := SELF.panes.Scanner;
001305          WHILE s.Scan(pane) DO
001306              BEGIN
001307              pane.GetScrollLimits(viewedLRect, scrollableLRect);
001308              WITH anLRect DO
001309                  BEGIN
001310                  LRectHaveLPt(scrollableLRect, topLeft);
001311                  LRectHaveLPt(scrollableLRect, botRight);
001312                  END;
001313
001314              WITH viewedLRect DO
001315                  IF top <= anLRect.bottom THEN
001316                      IF bottom >= anLRect.top THEN
001317                          IF left <= anLRect.right THEN
001318                              IF right >= anLRect.left THEN
001319                                  BEGIN
001320                                  s.Done;
001321                                  PaneShowing := pane;
001322                                  END;
001323              END;
001324          {$IFC fTrace}EP;{$ENDC}
001325      END;
001326
001327
001328      {$S sRes}
001329      FUNCTION  {TPanel.}PaneToScroll(VAR anLRect: LRect; hMinToSee, vMinToSee: INTEGER): TPane;
001330          VAR tempLRect:      LRect;
001331              pane:           TPane;
001332              dummyPt:        Point;
001333      BEGIN
001334          {$IFC fTrace}BP(5);{$ENDC}
001335          WITH anLRect DO
001336              BEGIN
001337              tempLRect.top := top + vMinToSee;
001338              tempLRect.bottom := bottom - vMinToSee;
001339              tempLRect.left := left + hMinToSee;
```

```
001340              tempLRect.right := right - hMinToSee;
001341              END;
001342
001343          pane := SELF.PaneShowing(tempLRect);
001344
001345          IF pane = NIL THEN
001346              BEGIN
001347              pane := SELF.PaneShowing(anLRect);
001348
001349              IF pane = NIL THEN
001350                  WITH SELF.lastClick DO
001351                      BEGIN
001352                      IF NOT gotPane THEN
001353                          BEGIN
001354                          {$H-}
001355                          clickPane := TPane(SELF.ChildWithPt(clickPt, SELF.panes, dummyPt));
001356                          {$H+}
001357                          gotPane := TRUE;
001358                          END;
001359                      pane := clickPane;
001360                      END;
001361              END
001362          ELSE
001363              pane := NIL; {already showing The Right Stuff}
001364
001365          PaneToScroll := pane;
001366          {$IFC fTrace}EP;{$ENDC}
001367      END;
001368
001369
001370      {$S sCldInit}
001371      PROCEDURE {TPanel.}Preview{(newMode: TPreviewMode)};
001372          VAR oldMode:            TPreviewMode;
001373              showMargins:        BOOLEAN;
001374              hideMargins:        BOOLEAN;
001375              noSelection:        TSelection;
001376              paginatedView:      TPaginatedView;
001377              vhs:                VHSelect;
001378              offset:             LPoint;
001379              bs:                 TListScanner;
001380              band:               TBand;
001381              firstPane:          TPane;
001382              pagiLPoint:         LPoint;
001383              ps:                 TListScanner;
001384              pane:               TPane;
001385              pageEditView:       TView;
001386              unPagLPt:           LPoint;     {and pageLocation out!}
001387
```

```
001388          PROCEDURE XorBreaksOnThePad;
001389          BEGIN
001390              SELF.view.printManager.DrawBreaks(FALSE);
001391          END;
001392
001393          PROCEDURE ClearSelection;
001394          BEGIN
001395              noSelection := SELF.selection.FreedAndReplacedBy(SELF.view.NoSelection);
001396          END;
001397
001398      BEGIN
001399          {$IFC fTrace}BP(9);{$ENDC}
001400          IF SELF.view.isPrintable THEN   {Actually shouldn't be called unless isPrintable}
001401              BEGIN
001402              oldMode := SELF.previewMode;
001403
001404              showMargins := (newMode = mPrvwMargins);
001405              hideMargins := (oldMode = mPrvwMargins);
001406
001407              IF oldMode = newMode THEN
001408                  BEGIN
001409                  END
001410              ELSE
001411              IF showMargins OR hideMargins  THEN
001412                  BEGIN
001413
001414                  paginatedView := SELF.paginatedView;
001415
001416                  IF showMargins THEN
001417                      BEGIN
001418                      paginatedView := SELF.view.printManager.NewPaginatedView(NIL);
001419                      SELF.currentView := paginatedView;
001420                      SELF.paginatedView := paginatedView;
001421                      END
001422                  ELSE
001423                      SELF.currentView := SELF.view; { newMode = show Breaks or show main view }
001424
001425                  SELF.previewMode := newMode;
001426
001427                  FOR vhs := v TO h DO
001428                      BEGIN
001429                      offset.vh[orthogonal[vhs]] := 0;
001430                      bs := SELF.bands[vhs].Scanner;
001431                      WHILE bs.Scan(band) DO
001432                          BEGIN
001433                          firstPane := TPane(band.panes.First);
001434                          IF showMargins THEN
001435                              paginatedView.PagifyLPoint(firstPane.viewedLRect.topLeft, pagiLPoint)
```

```
001436                          ELSE
001437                          IF hideMargins THEN
001438                              paginatedView.DePagifyLPoint(firstPane.viewedLRect.topLeft, pagiLPoint);
001439
001440                          offset.vh[vhs] := pagiLPoint.vh[vhs] - firstPane.viewedLRect.topLeft.vh[vhs];
001441                          ps := band.panes.Scanner;
001442                          WHILE ps.Scan(pane) DO
001443                              BEGIN
001444                              pane.currentView := SELF.currentView;
001445                              pane.OffsetBy(offset);
001446                              END;
001447                          END;
001448                      END;
001449
001450                  IF hideMargins THEN
001451                      BEGIN
001452                      paginatedView.Free;
001453                      SELF.paginatedView := NIL;
001454                      theMarginPad.view := NIL;
001455                      END;
001456
001457                  SELF.Rescroll;  {Does Invalidate and Moves Thumbs}
001458                  END
001459              ELSE
001460                  BEGIN
001461                  SELF.window.Update(TRUE);             {Update in the old mode, in case regions were invalid}
001462                  SELF.previewMode := newMode;    {Set the new mode}
001463                  SELF.OnAllPadsDo(XorBreaksOnThePad);    {Xor the page breaks}
001464                  END;
001465              END;
001466          {$IFC fTrace}EP;{$ENDC}
001467      END;
001468      {$S SgABCres}
001469
001470
001471     {$S SgABCpri}
001472          PROCEDURE {TPanel.}PrintView{(printPref: TPrReserve)};
001473      BEGIN
001474          {$IFC fTrace}BP(7);{$ENDC}
001475          IF SELF.view.printManager <> NIL THEN
001476              SELF.view.printManager.Print(printPref);
001477          {$IFC fTrace}EP;{$ENDC}
001478      END;
001479      {$S SgABCres}
001480
001481
001482  {$S sStartup}
001483      PROCEDURE {TPanel.}Refresh{(rActions: TActions; highTransit: THighTransit)};
```

```
001484
001485          PROCEDURE RefreshPane(obj: TObject);
001486              VAR pane:  TPane;
001487          BEGIN
001488              pane := TPane(obj);
001489              IF RectIsVisible(pane.outerRect) THEN
001490                  pane.Refresh(rActions, highTransit);
001491          END;
001492
001493      BEGIN
001494          {$IFC fTrace}BP(7);{$ENDC}
001495          {$IFC fDbgABC}
001496          IF (rBackground IN rActions) AND (highTransit > hOffToOn) THEN
001497              ABCBreak('Refresh: rBackground requested, but highTransit does not start from Off', 0);
001498          {$ENDC}
001499          IF rFrame IN rActions THEN
001500              SELF.Frame;
001501          SELF.panes.Each(RefreshPane);
001502          {$IFC fTrace}EP;{$ENDC}
001503      END;
001504  {$S SgABCres}
001505
001506
001507      {$S SgABCcld}
001508      PROCEDURE {TPanel.}RemakePanes;
001509          VAR vs, ps: TListScanner;
001510              band: TBand;
001511              pane: TPane;
001512      BEGIN
001513      {assumes they are right in the bands}
001514          {$IFC fTrace}BP(7);{$ENDC}
001515          SELF.panes.DelAll(FALSE);
001516          vs := SELF.bands[v].Scanner;
001517          WHILE vs.Scan(band) DO
001518              BEGIN
001519              ps := band.panes.Scanner;
001520              WHILE ps.Scan(pane) DO
001521                  SELF.panes.insLast(pane);
001522              END;
001523          {$IFC fTrace}EP;{$ENDC}
001524      END;
001525      {$S SgABCres}
001526
001527
001528      {$S SgABCcld}
001529      PROCEDURE {TPanel.}RememberSplit{(vhs: VHSelect; atCd: INTEGER)};
001530          VAR deletedSplits:  TArray;
001531      BEGIN
```

```
001532          {$IFC fTrace}BP(7);{$ENDC}
001533          deletedSplits := SELF.deletedSplits;
001534          IF deletedSplits <> NIL THEN
001535              BEGIN
001536              {$IFC fDbgABC}
001537              IF deletedSplits.recordBytes <> 2 THEN
001538                  ABCbreak('This panel has a deletedSplits array, but its recordBytes <> 2', ORD(SELF));
001539              {$ENDC}
001540
001541              IF vhs = v THEN
001542                  atCd := - atCd;
001543              deletedSplits.InsLast(@atCd);
001544              END;
001545          {$IFC fTrace}EP;{$ENDC}
001546      END;
001547      {$S SgABCres}
001548
001549
001550      {$S SgABCcld}
001551      PROCEDURE {TPanel.}Remove;
001552          VAR itsParent:      TBranchArea;
001553              itsGrandParent: TBranchArea;
001554              itsSibling:     TArea;
001555              itsWindow:      TWindow;
001556              firstPanel:     TPanel;
001557      BEGIN
001558          {$IFC fTrace}BP(7);{$ENDC}
001559          itsParent := SELF.parentBranch;
001560          itsWindow := SELF.window;
001561
001562          {$IFC fDbgABC}
001563          IF itsParent = NIL THEN
001564              ABCBreak('You cannot remove the last panel in the window', ORD(SELF));
001565          {$ENDC}
001566
001567          itsGrandParent := itsParent.parentBranch;
001568          itsSibling := itsParent.OtherChild(SELF);
001569
001570          itsSibling.ResizeOutside(itsParent.outerRect);
001571
001572          itsSibling.parentBranch := itsGrandParent;
001573          IF itsGrandParent = NIL THEN
001574              itsWindow.panelTree := itsSibling
001575          ELSE
001576              itsGrandParent.ReplaceChild(itsParent, itsSibling); {also sets my parentBranch to NIL}
001577
001578          SELF.resizeBranch := NIL;
001579
```

```
001580          firstPanel := TPanel(itsWindow.panels.First);
001581          IF itsWindow.selectPanel = SELF THEN
001582              itsWindow.selectPanel := firstPanel;
001583          IF itsWindow.clickPanel = SELF THEN
001584              itsWindow.clickPanel := firstPanel;
001585          {We do not change undoSelPanel & undoClickPanel because undo may bring them back; so caller beware!}
001586          itsParent.Free;
001587
001588          itsWindow.panels.DelObject(SELF, FALSE);
001589          {$IFC fTrace}EP;{$ENDC}
001590      END;
001591      {$S SgABCres}
001592
001593
001594      {$S SgABCcld}
001595      PROCEDURE {TPanel.}RepaneOrthogonalBands{(vhs: VHSelect)};
001596          VAR bs, orthoBs, ps: TListScanner;
001597              orthoBands:  TList;
001598              band, oBand: TBand;
001599              pane: TPane;
001600      BEGIN
001601      {assumes they are right in the orthogonal band}
001602          {$IFC fTrace}BP(7);{$ENDC}
001603          orthoBands := SELF.bands[orthogonal[vhs]];
001604          orthoBs := orthoBands.Scanner;
001605          while orthoBs.Scan(oBand) do
001606              oBand.panes.DelAll(FALSE);
001607          bs := SELF.bands[vhs].Scanner;
001608          WHILE bs.Scan(band) DO
001609              BEGIN
001610              ps := band.panes.Scanner;
001611              orthoBs := orthoBands.Scanner;
001612              WHILE ps.Scan(pane) AND orthoBs.Scan(oBand) DO
001613                  oBand.panes.insLast(pane);
001614              END;
001615          {$IFC fTrace}EP;{$ENDC}
001616      END;
001617      {$S SgABCres}
001618
001619
001620      {$S SgABCcld}
001621      PROCEDURE {TPanel.}Replace{(panel: TPanel)};
001622          VAR itsParent:      TBranchArea;
001623              itsWindow:      TWindow;
001624      BEGIN
001625          {$IFC fTrace}BP(7);{$ENDC}
001626          itsParent := SELF.parentBranch;
001627          itsWindow := SELF.window;
```

```
001628
001629          itsWindow.panels.DelObject(SELF, FALSE);
001630          itsWindow.panels.InsLast(panel);
001631
001632          IF itsParent = NIL THEN
001633              itsWindow.panelTree := panel
001634          ELSE
001635              itsParent.ReplaceChild(SELF, panel); {also sets my parentBranch to NIL}
001636
001637          SELF.resizeBranch := NIL;
001638
001639          panel.ResizeOutside(SELF.outerRect);
001640
001641          IF itsWindow.selectPanel = SELF THEN
001642              itsWindow.selectPanel := panel;
001643          IF itsWindow.clickPanel = SELF THEN
001644              itsWindow.clickPanel := panel;
001645          {We do not change undoSelPanel & undoClickPanel because undo may bring them back; so caller beware!}
001646          {$IFC fTrace}EP;{$ENDC}
001647      END;
001648      {$S SgABCres}
001649
001650
001651      {$S sCldInit}
001652      PROCEDURE {TPanel.}Rescroll;
001653          VAR vhs:    VHSelect;
001654              band:   TBand;
001655              s:      TListScanner;
001656      BEGIN
001657          {$IFC fTrace}BP(7);{$ENDC}
001658          InvalRect(SELF.outerRect);  {Since the view is changing, no part of the old image is good}
001659          FOR vhs := v TO h DO
001660              BEGIN
001661              s := SELF.bands[vhs].Scanner;
001662              WHILE s.Scan(band) DO
001663                  IF band.scroller <> NIL THEN
001664                      SetThumb(POINTER(band.scroller.sBoxID), band.ThumbPos);
001665                          {since we invalidated everything, just telling the SB library where the
001666                              thumb should be is enough}
001667              END;
001668          {$IFC fTrace}EP;{$ENDC}
001669      END;
001670      {$S SgABCres}
001671
001672
001673      {$S SgABCcld}
001674      PROCEDURE {TPanel.}ResizeBand{(vhs: VHSelect; band: TBand; newViewLCd: LONGINT;
001675                                      fInvalidate: BOOLEAN)};
```

```
001676          VAR scroller:       TScroller;
001677              sbRect:         Rect;
001678              tempRect:       Rect;
001679              toBeDeleted:    TList {OF TPane};
001680              ps:             TListScanner;
001681              pane:           TPane;
001682              oldBandInner:   Rect;
001683              newOuterRect:   Rect;
001684              unchangedRect:  Rect;
001685              tempBand:       TBand;
001686              sideBand:       TSideBand;
001687      BEGIN
001688          {$IFC fTrace}BP(7);{$ENDC}
001689          scroller := band.scroller;
001690
001691          IF scroller = NIL THEN {band is a side band}
001692              sbRect := band.outerRect
001693          ELSE
001694              BEGIN
001695              scroller.GetSize(sbRect);
001696              WITH sbRect DO {regular bands must lie within the contentRect; the +/- 1 is
001697                                  because the contentRect corresponds to the innerRect, but
001698                                  sbRect must be based on the outerRect}
001699                  BEGIN
001700                  topLeft.vh[vhs] := Max(topLeft.vh[vhs], SELF.contentRect.topLeft.vh[vhs] - 1);
001701                  botRight.vh[vhs] := Min(botRight.vh[vhs], SELF.contentRect.botRight.vh[vhs] + 1);
001702                  END;
001703              END;
001704
001705          unchangedRect := zeroRect;
001706
001707          IF LengthRect(sbRect, vhs) <= 0 THEN
001708              BEGIN
001709              toBeDeleted := TList.CREATE(NIL, SELF.Heap, band.panes.size);
001710              ps := band.panes.Scanner;
001711              WHILE ps.Scan(pane) DO
001712                  toBeDeleted.InsLast(pane);
001713              SELF.bands[vhs].DelObject(band, TRUE);
001714              SELF.CleanUpPanes(toBeDeleted);
001715              END
001716          ELSE
001717              BEGIN
001718              newOuterRect := SELF.innerRect;
001719              InsetRect(newOuterRect, -1, -1);
001720              AlignRect(newOuterRect, sbRect, vhs);
001721              oldBandInner := band.innerRect;
001722              band.SetOuterRect(newOuterRect);
001723              band.ResizePanes(newViewLCd);
```

```
001724
001725              IF fInvalidate THEN
001726                  IF band.ViewLCd = newViewLCd THEN
001727                      IF SectRect(oldBandInner, band.innerRect, unchangedRect) THEN;
001728              END;
001729
001730          IF fInvalidate THEN
001731              InvalDiffRect(band.outerRect, unchangedRect);
001732          {$IFC fTrace}EP;{$ENDC}
001733      END;
001734
001735
001736      {$S sCldInit}
001737      PROCEDURE {TPanel.}ResizeInside{(newInnerRect: Rect)};
001738          VAR toBeDeleted:        TList {OF TPane};
001739              allBandOuterRect:   Rect;
001740              vhs:                VHSelect;
001741              s:                  TListScanner;
001742              nextTopLeft:        INTEGER;
001743              lastBotRight:       INTEGER;
001744              thisBotRight:       INTEGER;
001745              band:               TBand;
001746              ps:                 TListScanner;
001747              pane:               TPane;
001748              newBandOuterRect:   Rect;
001749              oldViewLCd:         LONGINT;
001750              firstBand:          TBand;
001751              lastBand:           TBand;
001752      BEGIN
001753          {$IFC fTrace}BP(7);{$ENDC}
001754          toBeDeleted := TList.CREATE(NIL, SELF.Heap, 0);
001755          allBandOuterRect := newInnerRect;
001756          InsetRect(allBandOuterRect, -1, -1);
001757          FOR vhs := v TO h DO
001758              BEGIN
001759              firstBand := TBand(SELF.bands[vhs].First);
001760              IF firstBand.scroller = NIL THEN
001761                  firstBand := TSideBand(firstBand).CoBand;
001762              lastBand := TBand(SELF.bands[vhs].Last);
001763              IF lastBand.scroller = NIL THEN
001764                  lastBand := TSideBand(lastBand).CoBand;
001765
001766              {$H-}
001767              WITH SELF.contentRect DO
001768                  BEGIN
001769                  nextTopLeft := Max(topLeft.vh[vhs]-1, allBandOuterRect.topLeft.vh[vhs]);
001770                  lastBotRight := Min(botRight.vh[vhs]+1, allBandOuterRect.botRight.vh[vhs]);
001771                  END;
```

```
001772              {$H+}
001773
001774              s := SELF.bands[vhs].Scanner;
001775              WHILE s.Scan(band) DO
001776                  IF band.scroller = NIL THEN {a side band}
001777                      BEGIN
001778                      IF NOT TSideBand(band).topOrLeft THEN {a bottom/right side band must be moved
001779                                                             into a new position}
001780                          BEGIN
001781                          SELF.SideBandRect(vhs, FALSE, newBandOuterRect); {.SideBandRect returns an InnerRect}
001782                          InsetRect(newBandOuterRect, -1, -1); {outerRect is innerRect outset by 1...}
001783                          WITH newBandOuterRect.topLeft DO
001784                              vh[vhs] := vh[vhs] + 1;         {... EXCEPT on the top/left}
001785
001786                          band.ResizeOutside(newBandOuterRect);
001787                          END;
001788                      END
001789                  ELSE {a regular band}
001790                      {Always leave at least one pane}
001791                      IF (band <> firstBand) AND (nextTopLeft >= lastBotRight) THEN
001792                          BEGIN
001793                          ps := band.panes.Scanner;
001794                          WHILE ps.Scan(pane) DO
001795                              IF toBeDeleted.Pos(0, pane) <= 0 THEN
001796                                  toBeDeleted.InsLast(pane);
001797                          SELF.RememberSplit(vhs, band.outerRect.topLeft.vh[vhs]);
001798                          s.Delete(TRUE);
001799                          END
001800                      ELSE
001801                          BEGIN
001802                          newBandOuterRect.topLeft.vh[vhs] := nextTopLeft;
001803                          IF band = lastBand THEN
001804                              thisBotRight := lastBotRight
001805                          ELSE
001806                              thisBotRight := Min(nextTopLeft + lengthRect(band.outerRect, vhs), lastBotRight);
001807                          newBandOuterRect.botRight.vh[vhs] := thisBotRight;
001808                          AlignRect(newBandOuterRect, allBandOuterRect, orthogonal[vhs]);
001809                          oldViewLCd := band.ViewLCd;
001810                          band.ResizeOutside(newBandOuterRect);
001811                          IF oldViewLCd <> band.ViewLCd THEN
001812                              InvalRect(band.innerRect);
001813                          nextTopLeft := newBandOuterRect.botRight.vh[vhs];
001814                          END;
001815              END;
001816          SELF.CleanUpPanes(toBeDeleted);
001817          SELF.RestoreSplits; {do this after all the bands have been adjusted}
001818          {$IFC fTrace}EP;{$ENDC}
001819      END;
```

```
001820      {$S SgABCres}
001821
001822
001823      {$S SgABCcld}
001824      PROCEDURE {TPanel.}ResizeOutside{(newOuterRect: Rect)};
001825          VAR oldOuterRect:   Rect;
001826              oldInnerRect:   Rect;
001827              newInnerRect:   Rect;
001828              unchangedRect:  Rect;
001829
001830      BEGIN
001831          {$IFC fTrace}BP(7);{$ENDC}
001832          oldOuterRect := SELF.outerRect;
001833
001834          IF NOT EqualRect(oldOuterRect, newOuterRect) THEN
001835              BEGIN
001836              oldInnerRect := SELF.innerRect;
001837              SELF.DecideAboutBars(newOuterRect);
001838              newInnerRect := SELF.innerRect;
001839
001840              unchangedRect := zeroRect;
001841              IF EqualPt(oldOuterRect.topLeft, newOuterRect.topLeft) THEN
001842                  IF SectRect(oldInnerRect, newInnerRect, unchangedRect) THEN;
001843
001844              InvalDiffRect(newOuterRect, unchangedRect);
001845
001846              SELF.ResizeInside(newInnerRect);
001847              END;
001848          {$IFC fTrace}EP;{$ENDC}
001849      END;
001850      {$S SgABCres}
001851
001852
001853      {$S sCldInit}
001854      PROCEDURE {TPanel.}RestoreSplits;
001855          VAR deletedSplits:  TArray;
001856              contentRect:    Rect;
001857              vhs:            VHSelect;
001858              firstScrollers: ARRAY[VHSelect] OF TScroller;
001859              s:              TArrayScanner;
001860              pInt:           Ptr;
001861              cd:             INTEGER;
001862      BEGIN
001863          {$IFC fTrace}BP(7);{$ENDC}
001864          deletedSplits := SELF.deletedSplits;
001865
001866          IF deletedSplits <> NIL THEN
001867              BEGIN
```

```
001868              {$IFC fDbgABC}
001869              IF deletedSplits.recordBytes <> 2 THEN
001870                  ABCbreak('This panel has a deletedSplits array, but its recordBytes <> 2', ORD(SELF));
001871              {$ENDC}
001872
001873              contentRect := SELF.contentRect;
001874              FOR vhs := v TO h DO
001875                  firstScrollers[vhs] := SELF.scrollBars[vhs].firstBox;
001876
001877              s := deletedSplits.Scanner;
001878              WHILE s.Scan(pInt) DO
001879                  BEGIN
001880                  cd := TpInteger(pInt)^;
001881
001882                  IF cd < 0 THEN
001883                      BEGIN
001884                      vhs := v;
001885                      cd := - cd;
001886                      END
001887                  ELSE
001888                      vhs := h;
001889
001890                  IF (cd > contentRect.topLeft.vh[vhs]) AND
001891                     (cd < contentRect.botRight.vh[vhs] - dptSkewer.vh[vhs]) THEN
001892                      BEGIN
001893                      SELF.MoveSplitBefore(firstScrollers[vhs], cd);
001894                      s.Delete;
001895                      END;
001896                  END;
001897              END;
001898          {$IFC fTrace}EP;{$ENDC}
001899      END;
001900      {$S SgABCres}
001901
001902
001903      {$S sRes}
001904      PROCEDURE {TPanel.}RevealLRect(VAR anLRect: LRect; hMinToSee, vMinToSee: INTEGER);
001905          VAR pane:           TPane;
001906              revisedLRect:   LRect;
001907      BEGIN
001908          {$IFC fTrace}BP(5);{$ENDC}
001909          IF SELF.previewMode = mPrvwMargins THEN {need to map coords}
001910              BEGIN
001911              SELF.paginatedView.PagifyLPt(anLRect.topLeft, revisedLRect.topLeft);
001912              SELF.paginatedView.PagifyLPt(anLRect.botRight, revisedLRect.botRight);
001913              END
001914          ELSE
001915              revisedLRect := anLRect;
```

```
001916          pane := SELF.PaneToScroll(revisedLRect, hMinToSee, vMinToSee);
001917          IF pane <> NIL THEN
001918              pane.ScrollToReveal(revisedLRect, hMinToSee, vMinToSee);
001919          {$IFC fTrace}EP;{$ENDC}
001920      END;
001921
001922
001923      {$S sStartup}
001924      PROCEDURE {TPanel.}SetInnerRect{(newInnerRect: Rect)};
001925      BEGIN
001926          {$IFC fTrace}BP(11);{$ENDC}
001927          SUPERSELF.SetInnerRect(newInnerRect);
001928          SELF.ComputeContentRect;
001929          {$IFC fTrace}EP;{$ENDC}
001930      END;
001931
001932
001933      {$S sStartup}
001934      PROCEDURE {TPanel.}SetOuterRect{(newOuterRect: Rect)};
001935      BEGIN
001936          {$IFC fTrace}BP(11);{$ENDC}
001937          SUPERSELF.SetOuterRect(newOuterRect);
001938          SELF.ComputeContentRect;
001939          {$IFC fTrace}EP;{$ENDC}
001940      END;
001941
001942
001943      {$S SgABCcld}
001944      PROCEDURE {TPanel.}SetZoomFactor{(zoomNumerator, zoomDenominator: Point)};
001945          VAR s:      TListScanner;
001946              pane:   TPane;
001947      BEGIN
001948          {$IFC fTrace}BP(7);{$ENDC}
001949          Reduce(zoomNumerator.h, zoomDenominator.h); {reduce to lowest terms}
001950          Reduce(zoomNumerator.v, zoomDenominator.v); {reduce to lowest terms}
001951          {$IFC fDbgABC}
001952          IF fExperimenting THEN
001953              WriteLn('New (h) Zoom: ', zoomNumerator.h:1, '/', zoomDenominator.h:1);
001954          {$ENDC}
001955          WITH SELF, zoomFactor DO
001956              BEGIN
001957              numerator := zoomNumerator;
001958              denominator := zoomDenominator;
001959              zoomed := (numerator.h <> denominator.h) OR (numerator.v <> denominator.v);
001960              END;
001961          s := SELF.panes.Scanner;
001962          WHILE s.Scan(pane) DO
001963              pane.SetZoomFactor(zoomNumerator, zoomDenominator);
```

```
001964          SELF.Rescroll;  {Does Invalidate and Moves Thumbs}
001965          {$IFC fTrace}EP;{$ENDC}
001966      END;
001967      {$S SgABCres}
001968
001969
001970  {$S SgABCcld}
001971      PROCEDURE {TPanel.}ShowSideBand{(vhs: VHSelect; topOrLeft: BOOLEAN; size: INTEGER; viewLCd: LONGINT)};
001972          VAR x:              INTEGER;
001973              bandIndex:      INTEGER;
001974              band:           TBand;
001975              contentRect:    Rect;
001976              tempRect:       Rect;
001977              oldSideSize:    INTEGER;
001978              newViewLCd:     LONGINT;
001979              scroller:       TScroller;
001980              s:              TListScanner;
001981              bandVHS:        VHSelect;
001982              bandIsCovered:  BOOLEAN;
001983              moveNextSplit:  BOOLEAN;
001984              removeCd:       INTEGER;
001985              coBand:         TBand;
001986
001987      BEGIN
001988          {$IFC fTrace}BP(12);{$ENDC}
001989          SELF.SideBandRect(vhs, topOrLeft, tempRect);
001990          oldSideSize := LengthRect(tempRect, vhs);
001991          x := Max(-1, size);
001992
001993          WITH SELF DO
001994              IF topOrLeft THEN
001995                  tlSideBandSize.vh[vhs] := x
001996              ELSE
001997                  brSideBandSize.vh[vhs] := x;
001998
001999          SELF.ComputeContentRect;
002000
002001          IF size > oldSideSize THEN
002002              BEGIN
002003              SELF.window.Resize(FALSE); {make sure we have enough space for the bigger side band}
002004
002005              { delete splits that are now covered by the bigger side band }
002006              IF topOrLeft THEN
002007                  removeCd := 0
002008              ELSE
002009                  removeCd := MAXINT;
002010
002011              moveNextSplit := FALSE;
```

```
002012              s := SELF.bands[vhs].Scanner;
002013
002014              WHILE s.Scan(band) DO
002015                  IF band.scroller <> NIL THEN {not a side band}
002016                      BEGIN
002017                      bandIsCovered := NOT SectRect(band.innerRect, SELF.contentRect, tempRect);
002018
002019                      IF bandIsCovered THEN
002020                          s.Delete(FALSE); {delete it from the list before some other method, so our scanner
002021                                            doesn't get confused; it will still get freed later, though}
002022
002023                      IF moveNextSplit OR (bandIsConvered AND NOT topOrLeft) THEN
002024                          BEGIN
002025                          SELF.RememberSplit(vhs, band.outerRect.topLeft.vh[vhs]);
002026                          SELF.MoveSplitBefore(band.scroller, removeCd);
002027                          END;
002028
002029                      moveNextSplit := bandIsCovered AND topOrLeft;
002030                      END;
002031              END;
002032
002033          SELF.SideBandRect(vhs, topOrLeft, tempRect);
002034
002035          {Create/Resize/Delete the sideBand}
002036          IF (oldSideSize = -1) AND (size >= 0) THEN {create}
002037              BEGIN
002038              band := TSideBand.CREATE(NIL, SELF.Heap, SELF, tempRect, vhs, topOrLeft, viewLCd);
002039              coBand := TSideBand(band).Coband;
002040
002041              InvalRect(tempRect);
002042
002043              SELF.RepaneOrthogonalBands(vhs);
002044              SELF.RemakePanes;
002045
002046              {calculate the new viewLCd for the side band's coBand}
002047              newViewLCd := coBand.ViewLCd;
002048              IF topOrLeft THEN
002049                  newViewLCd := newViewLCd + size + 1;
002050              END
002051          ELSE IF oldSideSize >= 0 THEN
002052              BEGIN
002053              IF topOrLeft THEN {get the side band to resize into band}
002054                  band := TBand(SELF.bands[vhs].First)
002055              ELSE
002056                  band := TBand(SELF.bands[vhs].Last);
002057              coband := TSideBand(band).Coband;
002058
002059              band.SetInnerRect(tempRect); {side bands are resized according to their current inner/outerRects}
```

```
002060              SELF.ResizeBand(vhs, band, band.ViewLCd, TRUE);
002061
002062              {calculate the new viewLCd for the side band's coBand}
002063              newViewLCd := coBand.ViewLCd;
002064              IF topOrLeft THEN
002065                  newViewLCd := newViewLCd + size - oldSideSize;
002066              END
002067          ELSE
002068              coBand := NIL;
002069
002070          IF coBand <> NIL THEN
002071              BEGIN
002072              {resize the regular band that is next to the sideBand (coband)}
002073              SELF.ResizeBand(vhs, coBand, newViewLCd, TRUE);
002074
002075              {invalidate the scroller associated with coBand}
002076              coBand.scroller.GetSize(tempRect);
002077              InvalRect(tempRect);
002078
002079              SELF.RestoreSplits;
002080              END;
002081          {$IFC fTrace}EP;{$ENDC}
002082      END;
002083  {$S SgABCres}
002084
002085
002086      {$S SgABCcld}
002087      PROCEDURE {TPanel.}SideBandRect{(vhs: VHSelect; topOrLeft: BOOLEAN; VAR bandRect: Rect)};
002088          {gets the innerRect of a side band, given the current contentRect}
002089          VAR contentRect:    Rect;
002090      BEGIN
002091          bandRect := SELF.innerRect;
002092          WITH bandRect DO
002093              IF topOrLeft THEN
002094                  botRight.vh[vhs] := topLeft.vh[vhs] + SELF.tlSideBandSize.vh[vhs]
002095              ELSE
002096                  topLeft.vh[vhs] := botRight.vh[vhs] - SELF.brSideBandSize.vh[vhs];
002097      END;
002098
002099
002100  {$S SgABCini}
002101  END;
002102  {$S SgABCres}
002103
002104
002105  METHODS OF TBand;
002106
002107
```

```
002108      {$S sCldInit}
002109      FUNCTION  {TBand.}CREATE{(object: TObject; heap: THeap; itsPanel: TPanel; itsInnerRect: Rect;
002110                            itsScroller: TScroller; itsDir: VHSelect): TBand};
002111          VAR panes: TList;
002112      BEGIN
002113          {$IFC fTrace}BP(7);{$ENDC}
002114          IF object = NIL THEN
002115              object := NewObject(heap, THISCLASS);
002116          SELF := TBand(object);
002117
002118          WITH SELF DO
002119              BEGIN
002120              window := itsPanel.window;
002121              panel := itsPanel;
002122              scroller := itsScroller;
002123              scrollDir := itsDir;
002124              parentBranch := NIL;
002125              END;
002126          panes := TList.CREATE(NIL, heap, 1);
002127          SELF.panes := panes;
002128          SELF.SetInnerRect(itsInnerRect);
002129          IF itsScroller <> NIL THEN
002130              itsScroller.band := SELF;
002131          {$IFC fTrace}EP;{$ENDC}
002132      END;
002133      {$S SgABCres}
002134
002135
002136      {$S SgABCini}
002137      PROCEDURE {TBand.}Free;
002138      BEGIN
002139          {$IFC fTrace}BP(7);{$ENDC}
002140          Free(SELF.scroller);
002141          SELF.panes.FreeObject;
002142          TArea.Free;
002143          {$IFC fTrace}EP;{$ENDC}
002144      END;
002145      {$S SgABCres}
002146
002147
002148      {$IFC fDebugMethods}
002149      {$S SgABCdbg}
002150      PROCEDURE {TBand.}Fields{(PROCEDURE Field(nameAndType: S255))};
002151      BEGIN
002152          TArea.Fields(Field);
002153          Field('window: TWindow');
002154          Field('panes: TList');
002155          Field('panel: TPanel');
```

```
002156          Field('scroller: TScroller');
002157          Field('scrollDir: Byte');
002158          Field('');
002159      END;
002160      {$S SgABCres}
002161      {$ENDC}
002162
002163
002164      {$S sScroll}
002165      PROCEDURE {TBand.}OffsetPanes{(deltaLPt: LPoint)};
002166
002167          PROCEDURE YouOffset(obj: TObject);
002168          BEGIN
002169              TPane(obj).OffsetBy(deltaLPt);
002170          END;
002171
002172      BEGIN
002173          {$IFC fTrace}BP(7);{$ENDC}
002174          SELF.panes.Each(YouOffset);
002175          {$IFC fTrace}EP;{$ENDC}
002176      END;
002177
002178
002179      {$S sCldInit}
002180      PROCEDURE {TBand.}ResizeOutside{(newOuterRect: Rect)};
002181          VAR scroller:           TScroller;
002182              newScrollerSize:    Rect;
002183              unchangedRect:      Rect;
002184      BEGIN
002185          {$IFC fTrace}BP(7);{$ENDC}
002186          IF NOT EqualRect(SELF.outerRect, newOuterRect) THEN
002187              BEGIN
002188              unchangedRect := SELF.outerRect;
002189              IF NOT EqualPt(unchangedRect.topLeft, newOuterRect.topLeft) THEN
002190                  unchangedRect := zeroRect
002191              ELSE
002192                  InsetRect(unchangedRect, 1, 1); {we want unchangedRect to be the old innerRect}
002193              InvalDiffRect(newOuterRect, unchangedRect);
002194
002195              scroller := SELF.scroller;
002196              SELF.SetOuterRect(newOuterRect);
002197
002198              newScrollerSize := SELF.outerRect;
002199              WITH SELF DO
002200                  BEGIN
002201                  newScrollerSize.botRight.vh[orthogonal[scrollDir]] :=
002202                      panel.innerRect.botRight.vh[orthogonal[scrollDir]] + 1;
002203
```

```
002204                  IF innerRect.topLeft.vh[scrollDir] = panel.contentRect.topLeft.vh[scrollDir] THEN
002205                      newScrollerSize.topLeft.vh[scrollDir] := panel.innerRect.topLeft.vh[scrollDir] - 1;
002206
002207                  IF innerRect.botRight.vh[scrollDir] = panel.contentRect.botRight.vh[scrollDir] THEN
002208                      newScrollerSize.botRight.vh[scrollDir] := panel.innerRect.botRight.vh[scrollDir] + 1;
002209                  END;
002210              scroller.SetSize(newScrollerSize);
002211
002212              SELF.ResizePanes(SELF.ViewLCd);
002213              END;
002214          {$IFC fTrace}EP;{$ENDC}
002215      END;
002216      {$S SgABCres}
002217
002218
002219      {$S sCldInit}
002220      PROCEDURE {TBand.}ResizePanes{(newViewLCd: LONGINT)};
002221      {assumes SELF.innerRect already set}
002222          VAR vhs:              VHSelect;
002223              s:                TListScanner;
002224              pane:             TPane;
002225              viewedLRect:      LRect;
002226              scrollableLRect:  LRect;
002227              oldViewLCd:       LONGINT;
002228              deltaLPt:         LPoint;
002229      BEGIN
002230          {$IFC fTrace}BP(7);{$ENDC}
002231          vhs := SELF.scrollDir;
002232
002233          s := SELF.panes.Scanner;
002234          WHILE s.Scan(pane) DO
002235              pane.Resize(SELF.innerRect, vhs);
002236
002237          IF SELF.panel.currentView <> NIL THEN
002238              BEGIN
002239              pane := TPane(SELF.panes.First);
002240              pane.GetScrollLimits(viewedLRect, scrollableLRect);
002241              oldViewLCd := SELF.ViewLCd;
002242              newViewLCd := Max(scrollableLRect.topLeft.vh[vhs],
002243                                Min(scrollableLRect.botRight.vh[vhs] - LengthLRect(viewedLRect, vhs),
002244                                    newViewLCd));
002245
002246              deltaLPt.vh[orthogonal[vhs]] := 0;
002247        {$H-} deltaLPt.vh[vhs] := newViewLCd - oldViewLCd; {$H+}
002248              SELF.OffsetPanes(deltaLPt);
002249              SELF.ScrollBy(0);
002250              SetThumb(POINTER(SELF.scroller.sBoxID), SELF.ThumbPos);
002251                  {need to set thumb because band changed size}
```

```
002252              END;
002253          {$IFC fTrace}EP;{$ENDC}
002254      END;
002255      {$S SgABCres}
002256
002257
002258      {$S sScroll}
002259      PROCEDURE {TBand.}ScrollBy{(deltaLCd: LONGINT)};
002260          {positive scrolls towards end; 0 means resize & don't move thumb}
002261          VAR deltaLPt:   LPoint;
002262              vhs:        VHSelect;
002263      BEGIN
002264          {$IFC fTrace}BP(6);{$ENDC}
002265          PushFocus;
002266          SELF.window.Focus;
002267
002268          WITH SELF, deltaLPt DO
002269             BEGIN
002270             vhs := scrollDir;
002271             vh[vhs] := deltaLCd;
002272             vh[orthogonal[vhs]] := 0;
002273             END;
002274
002275          SELF.panel.DoScrolling(SELF, TPane(SELF.panes.First), vhs=h, vhs=v, deltaLPt);
002276
002277          IF NOT EqualLPt(deltaLPt, zeroLPt) THEN
002278              SELF.OffsetPanes(deltaLPt);
002279
002280          IF deltaLCd <> 0 THEN
002281              IF SELF.scroller <> NIL THEN {can this be a side band???}
002282                  SELF.scroller.MoveThumb(SELF.ThumbPos);
002283
002284          PopFocus;
002285          {$IFC fTrace}EP;{$ENDC}
002286      END;
002287
002288
002289      {$S sScroll}
002290      PROCEDURE {TBand.}ScrollStep{(icon: TEnumIcons; deltaLStd: LONGINT)};
002291          VAR vhs:        VHSelect;
002292              len:        LONGINT;
002293              deltaLCd:   LONGINT;
002294      BEGIN
002295          {$IFC fTrace}BP(6);{$ENDC}
002296          vhs := SELF.scrollDir;
002297          len := LIntDivInt(LengthRect(SELF.innerRect, vhs) * ORD4(SELF.panel.view.res.vh[vhs]),
002298                          screenRes.vh[vhs]);
002299
```

```
002300          CASE icon OF  {how far to scroll without regard for overshooting the ends}
002301              iScrollBack:
002302                  deltaLCd := -deltaLStd;
002303              iScrollFwd:
002304                  deltaLCd := deltaLStd;
002305              iFlipBack:
002306                  deltaLCd := Min(deltaLStd - len, -deltaLStd);
002307              iFlipFwd:
002308                  deltaLCd := Max(len - deltaLStd, deltaLStd);
002309              END;
002310          SELF.ScrollBy(deltaLCd);
002311          {$IFC fTrace}EP;{$ENDC}
002312      END;
002313
002314
002315      {$S sScroll}
002316      PROCEDURE {TBand.}ScrollTo{(viewLCd: LONGINT)};
002317          VAR pane:       TPane;
002318              deltaLCd:   LONGINT;
002319      BEGIN
002320          {$IFC fTrace}BP(6);{$ENDC}
002321          pane := TPane(SELF.panes.First);
002322          deltaLCd := viewLCd  - pane.viewedLRect.topLeft.vh[SELF.scrollDir];
002323          SELF.ScrollBy(deltaLCd);
002324          {$IFC fTrace}EP;{$ENDC}
002325      END;
002326
002327
002328      {$S sScroll}
002329      FUNCTION  {TBand.}ThumbPos{: INTEGER};
002330          VAR vhs:                VHSelect;
002331              pane:               TPane;
002332              viewedLRect:        LRect;
002333              scrollableLRect:    LRect;
002334              thumbLRange:        LONGINT;
002335              barRange:           INTEGER;
002336              lOffset:            LONGINT;
002337              barPos:             INTEGER;
002338      BEGIN
002339          {$IFC fTrace}BP(4);{$ENDC}
002340          vhs := SELF.scrollDir;
002341          pane := TPane(SELF.panes.First);
002342          pane.GetScrollLimits(viewedLRect, scrollableLRect);
002343          thumbLRange := LengthLRect(scrollableLRect, vhs) - LengthLRect(viewedLRect, vhs);
002344          barRange := SELF.scroller.ThumbRange;
002345
002346          IF barRange = 0 THEN
002347              ThumbPos := 0
```

```
002348          ELSE
002349              BEGIN
002350              lOffset := viewedLRect.topLeft.vh[vhs] - scrollableLRect.topLeft.vh[vhs];
002351
002352              IF thumbLRange > 1 THEN                         {Only divide by positive denominators}
002353                  barPos := LIntDivLInt(LIntMulInt(lOffset, barRange - 1) + thumbLRange - barRange,
002354                                                  thumbLRange - 1)
002355              ELSE
002356              IF (thumbLRange = 1) AND (lOffset > 0) THEN     {Very rare case: view one pixel bigger
002357                                                               than pane...}
002358                  barPos := barRange                          {...and scrolled to end}
002359              ELSE
002360                  barPos := 0;                                {Usually because the view is smaller than the
002361                                                               pane}
002362              {barPos = 0 or barRange only if nowhere to scroll}
002363
002364              ThumbPos := Max(0, Min(1000, LIntDivInt(LIntMulInt(barPos, 1000) + barRange - 1, barRange)));
002365                {ThumbPos = 0 or 1000 only if nowhere to scroll [assumes band is <= 1000 pixels long]}
002366              END;
002367          {$IFC fTrace}EP;{$ENDC}
002368      END;
002369
002370
002371      {$S sScroll}
002372      PROCEDURE {TBand.}ThumbTo{(newThumbPos: INTEGER)};
002373          VAR vhs:                VHSelect;
002374              thumbLRange:        LONGINT;
002375              pane:               TPane;
002376              viewedLRect:        LRect;
002377              scrollableLRect:    LRect;
002378      BEGIN
002379          {$IFC fTrace}BP(6);{$ENDC}
002380          vhs := SELF.scrollDir;
002381          pane := TPane(SELF.panes.First);
002382          pane.GetScrollLimits(viewedLRect, scrollableLRect);
002383          thumbLRange := LengthLRect(scrollableLRect, vhs) - LengthLRect(viewedLRect, vhs);
002384          SELF.ScrollTo(scrollableLRect.topLeft.vh[vhs] +
002385                          LIntDivInt(LIntMulInt(thumbLRange, newThumbPos), 1000));
002386          {$IFC fTrace}EP;{$ENDC}
002387      END;
002388
002389
002390      {$S sRes}
002391      FUNCTION  {TBand.}ViewLCd{: LONGINT};
002392          VAR pane:   TPane;
002393      BEGIN
002394          {$IFC fTrace}BP(4);{$ENDC}
002395          pane := TPane(SELF.panes.First);
```

```
002396          ViewLCd := pane.viewedLRect.topLeft.vh[SELF.scrollDir];
002397          {$IFC fTrace}EP;{$ENDC}
002398      END;
002399
002400
002401  {$S SgABCini}
002402  END;
002403  {$S SgABCres}
002404
002405
002406
002407  METHODS OF TSideBand;
002408
002409      {$S SgABCcld}
002410      FUNCTION  {TSideBand.}CREATE{(object: TObject; heap: THeap; itsPanel: TPanel; itsInnerRect: Rect;
002411                            itsDir: VHSelect; itsTopOrLeft: BOOLEAN;
002412                            itsViewLCd: LONGINT): TSideBand};
002413      VAR bandList:   TList;
002414          itsCoBand:  TBand;
002415          deltaLPt:   LPoint;
002416      BEGIN
002417          {$IFC fTrace}BP(7);{$ENDC}
002418          IF object = NIL THEN
002419              object := NewObject(heap, THISCLASS);
002420          TSideBand(object).topOrLeft := itsTopOrLeft; {needed to be set before SetInnerRect, which is
002421                                                  done in TBand.CREATE}
002422
002423          SELF := TSideBand(TBand.CREATE(object, heap, itsPanel, itsInnerRect, NIL, itsDir));
002424
002425          bandList := itsPanel.bands[itsDir];
002426
002427          IF itsTopOrLeft THEN
002428              BEGIN
002429              itsCoBand := TBand(bandList.First);
002430              bandList.InsFirst(SELF);
002431              END
002432          ELSE
002433              BEGIN
002434              itsCoBand := TBand(bandList.Last);
002435              bandList.InsLast(SELF);
002436              END;
002437
002438          SELF.panes.Become(itsCoBand.panes.Clone(heap));
002439          SELF.ResizePanes(itsViewLCd);
002440          {$IFC fTrace}EP;{$ENDC}
002441      END;
002442
002443
```

```
002444      {$S SgABCcld}
002445      PROCEDURE {TSideBand.}Free;
002446      BEGIN
002447          {$IFC fTrace}BP(3);{$ENDC}
002448          SELF.scroller := NIL;   {let my coBand free the scroller}
002449          SUPERSELF.Free;
002450          {$IFC fTrace}EP;{$ENDC}
002451      END;
002452
002453      {$IFC fDebugMethods}
002454      {$S SgABCdbg}
002455      PROCEDURE {TSideBand.}Fields{(PROCEDURE Field(nameAndType: S255))};
002456      BEGIN
002457          {$IFC fTrace}BP(10);{$ENDC}
002458          SUPERSELF.Fields(Field);
002459          Field('topOrLeft: BOOLEAN');
002460          Field('');
002461          {$IFC fTrace}EP;{$ENDC}
002462      END;
002463      {$ENDC}
002464
002465
002466      {$S SgABCcld}
002467      FUNCTION  {TSideBand.}CoBand{: TBand};
002468          VAR bandList:   TList;
002469      BEGIN
002470          {$IFC fTrace}BP(7);{$ENDC}
002471          bandList := SELF.panel.bands[SELF.scrollDir];
002472          IF SELF.topOrLeft THEN
002473              CoBand := TBand(bandList.At(2))
002474          ELSE
002475              CoBand := TBand(bandList.At(bandList.Size-1));
002476          {$IFC fTrace}EP;{$ENDC}
002477      END;
002478
002479
002480      {$S SgABCcld}
002481      PROCEDURE {TSideBand.}GetBorder{(VAR border: Rect)};
002482      BEGIN
002483          {$IFC fTrace}BP(7);{$ENDC}
002484          SUPERSELF.GetBorder(border);
002485          WITH SELF, border DO
002486              IF topOrLeft THEN
002487                  botRight.vh[scrollDir] := 0
002488              ELSE
002489                  topLeft.vh[scrollDir] := 0;
002490          {$IFC fTrace}EP;{$ENDC}
002491      END;
```

```
002492
002493
002494      {$S SgABCcld}
002495      PROCEDURE {TSideBand.}ResizeOutside{(newOuterRect: Rect)};
002496          VAR unchangedRect:      Rect;
002497              rectToInval:        Rect;
002498      BEGIN
002499          {$IFC fTrace}BP(7);{$ENDC}
002500          IF NOT EqualRect(SELF.outerRect, newOuterRect) THEN
002501              BEGIN
002502              unchangedRect := SELF.outerRect;
002503              IF NOT EqualPt(unchangedRect.topLeft, newOuterRect.topLeft) THEN
002504                  unchangedRect := zeroRect
002505              ELSE
002506                  InsetRect(unchangedRect, 1, 1); {we want unchangedRect to be the old innerRect}
002507
002508              SELF.SetOuterRect(newOuterRect);
002509
002510              rectToInval := SELF.innerRect;
002511              InsetRect(rectToInval, -1, -1);
002512              InvalDiffRect(rectToInval, unchangedRect);
002513
002514              SELF.ResizePanes(SELF.ViewLCd);
002515              END;
002516          {$IFC fTrace}EP;{$ENDC}
002517      END;
002518
002519
002520      {$S SgABCcld}
002521      PROCEDURE {TSideBand.}ResizePanes{(newViewLCd: LONGINT)};
002522      {assumes SELF.innerRect already set}
002523          VAR vhs:                VHSelect;
002524              s:                  TListScanner;
002525              pane:               TPane;
002526              viewedLRect:        LRect;
002527              scrollableLRect:    LRect;
002528              oldViewLCd:         LONGINT;
002529              deltaLPt:           LPoint;
002530      BEGIN
002531          {$IFC fTrace}BP(7);{$ENDC}
002532          vhs := SELF.scrollDir;
002533
002534          s := SELF.panes.Scanner;
002535          WHILE s.Scan(pane) DO
002536              pane.Resize(SELF.innerRect, vhs);
002537
002538          oldViewLCd := SELF.ViewLCd;
002539
```

```
002540          deltaLPt.vh[orthogonal[vhs]] := 0;
002541          deltaLPt.vh[vhs] := newViewLCd - oldViewLCd;
002542          SELF.OffsetPanes(deltaLPt);
002543
002544          {$IFC fTrace}EP;{$ENDC}
002545      END;
002546
002547
002548      {$S sError}
002549      PROCEDURE {TSideBand.}ScrollTo{(viewLCd: LONGINT)};
002550      BEGIN
002551          {$IFC fTrace}BP(6);{$ENDC}
002552          ABCBreak('Can not do TSideBand.ScrollTo', 0);
002553          {$IFC fTrace}EP;{$ENDC}
002554      END;
002555
002556
002557      {$S SgABCcld}
002558      FUNCTION  {TSideBand.}ThumbPos{: INTEGER};
002559      BEGIN
002560          {$IFC fTrace}BP(4);{$ENDC}
002561          ThumbPos := 0;
002562          {$IFC fTrace}EP;{$ENDC}
002563      END;
002564
002565
002566  {$S SgABCini}
002567  END;
002568  {$S SgABCres}
002569
002570
002571
002572  METHODS OF TPane;
002573
002574
002575      {$S sCldInit}
002576      FUNCTION  {TPane.}CREATE{(object: TObject; heap: THeap; itsPanel: TPanel; itsInnerRect: Rect;
002577                                itsViewedLRect: LRect): TPane};
002578      BEGIN
002579          {$IFC fTrace}BP(7);{$ENDC}
002580          IF object = NIL THEN
002581              object := NewObject(heap, THISCLASS);
002582          SELF := TPane(TPad.CREATE(object, heap, itsInnerRect, itsViewedLRect, screenRes,
002583                                    screenRes, POINTER(itsPanel.window.wmgrId)));
002584
002585          SELF.currentView := itsPanel.currentView;  {presumably unnecessary because will be done by haveView}
002586          SELF.panel := itsPanel;
002587          {$IFC fTrace}EP;{$ENDC}
```

```
002588      END;
002589      {$S SgABCres}
002590
002591
002592      {$IFC fDebugMethods}
002593      {$S SgABCdbg}
002594      PROCEDURE {TPane.}Fields{(PROCEDURE Field(nameAndType: S255))};
002595      BEGIN
002596          TPad.Fields(Field);
002597          Field('currentView: TView');
002598          Field('panel: TPanel');
002599      END;
002600      {$S SgABCres}
002601      {$ENDC}
002602
002603
002604      {$S sRes}
002605      FUNCTION  {TPane.}CursorAt{(mousePt: Point): TCursorNumber};
002606          {assumes mousePt is within the pane's innerRect}
002607          VAR mouseLPt:   LPoint;
002608              panePt:     Point; {window-relative, under the coordinate system defined by pane}
002609      BEGIN
002610          {$IFC fTrace}BP(2);{$ENDC}
002611          PushFocus;
002612          panePt := mousePt;
002613          LocalToGlobal(panePt);
002614          SELF.Focus;
002615          GlobalToLocal(panePt); {mousePt is now adjusted for the pane's new origin}
002616          SELF.PtToLPt(panePt, mouseLPt);
002617          IF LRectHasLPt(SELF.currentView.extentLRect, mouseLPt) THEN
002618              CursorAt := SELF.currentView.CursorAt(mouseLPt)
002619          ELSE
002620              CursorAt := arrowCursor;
002621          PopFocus;
002622          {$IFC fTrace}EP;{$ENDC}
002623      END;
002624
002625
002626      {$S sRes}
002627      PROCEDURE {TPane.}GetScrollLimits{(VAR viewedLRect, scrollableLRect: LRect)};
002628          VAR extra:  Point;
002629      BEGIN
002630          {$IFC fTrace}BP(3);{$ENDC}
002631          viewedLRect := SELF.viewedLRect;
002632          WITH SELF.currentView DO
002633              BEGIN
002634              scrollableLRect := extentLRect;
002635              extra := scrollPastEnd;
```

```
002636              END;
002637
002638          WITH scrollableLRect, extra DO
002639              BEGIN
002640              right  := right + Max(0, Min(viewedLRect.right - viewedLRect.left - h, h));
002641              bottom := bottom + Max(0, Min(viewedLRect.bottom - viewedLRect.top - v, v));
002642              END;
002643          {$IFC fTrace}EP;{$ENDC}
002644      END;
002645
002646
002647
002648      {$S sCldInit}
002649      PROCEDURE {TPane.}HaveView{(view: TView)};
002650          VAR deltaLPt:       LPoint;
002651              viewedLRect:    LRect;
002652              paneSize:       Point;
002653      BEGIN
002654          {$IFC fTrace}BP(7);{$ENDC}
002655          SELF.currentView := view;
002656          IF (view.res.h <> SELF.viewedRes.h) OR (view.res.v <> SELF.viewedRes.v) THEN
002657              BEGIN
002658              PtMinusPt(SELF.innerRect.botRight, SELF.innerRect.topLeft, paneSize);
002659              viewedLRect := view.extentLRect;
002660              viewedLRect.right := viewedLRect.left +
002661                  LIntDivInt(ORD4(paneSize.h) * view.res.h, SELF.padRes.h);
002662              viewedLRect.bottom := viewedLRect.top +
002663                  LIntDivInt(ORD4(paneSize.v) * view.res.v, SELF.padRes.v);
002664              SELF.Redefine(SELF.innerRect, viewedLRect, SELF.padRes, view.res, SELF.zoomFactor, SELF.port);
002665              END
002666          ELSE
002667              BEGIN
002668              SetLPt(deltaLPt, view.extentLRect.left - SELF.viewedLRect.left,
002669                               view.extentLRect.top - SELF.viewedLRect.top);
002670              SELF.OffsetBy(deltaLPt);
002671              END;
002672          {$IFC fTrace}EP;{$ENDC}
002673      END;
002674      {$S SgABCres}
002675
002676
002677      {$S sRes}
002678      PROCEDURE {TPane.}MouseTrack{(mPhase: TMousePhase; mousePt: Point)};
002679          {assumes mousePt is within the pane's innerRect;
002680              mousePt is window-relative, (0,0)-origined}
002681          VAR mouseLPt:       LPoint;
002682              panePt:         Point; {window-relative, under the coordinate system defined by pane}
002683              currentView:    TView;
```

```
002684      BEGIN
002685          {$IFC fTrace}BP(7);{$ENDC}
002686          PushFocus;
002687
002688          panePt := mousePt;
002689          LocalToGlobal(panePt);
002690          SELF.Focus;
002691          GlobalToLocal(panePt); {mousePt is now adjusted for the pane's new origin}
002692          SELF.PtToLPt(panePt, mouseLPt);
002693          currentView := SELF.currentView;
002694          currentView.MouseTrack(mPhase, mouseLPt);
002695          PopFocus;
002696
002697          { &&& we should optimize the following -- SELF.CursorAt also does the same focusing as above }
002698          process.ChangeCursor(SELF.CursorAt(mousePt));
002699          {$IFC fTrace}EP;{$ENDC}
002700      END;
002701
002702  {$S sStartup}
002703  {+++LSR+++} {This whole method is substantially changed}
002704      PROCEDURE {TPane.}Refresh{(rActions: TActions; highTransit: THighTransit)};
002705
002706          VAR panel:              TPanel;
002707              needGray:           BOOLEAN;
002708              viewExtentLRect:    LRect;
002709              viewedLRect:        LRect;
002710              tempLRect:          LRect;
002711
002712          PROCEDURE HighlightOnThePad;
002713          BEGIN
002714              panel.selection.Highlight(highTransit);
002715          END;
002716
002717      BEGIN
002718          {$IFC fTrace}BP(7);{$ENDC}
002719          panel := SELF.panel;
002720
002721          viewExtentLRect := SELF.currentView.extentLRect;
002722          viewedLRect := SELF.viewedLRect;
002723
002724          IF rFrame IN rActions THEN
002725              SELF.Frame;
002726
002727          needGray := (rBackground IN rActions) AND
002728                      ((viewedLRect.right > viewExtentLRect.right) OR
002729                       (viewedLRect.bottom > viewExtentLRect.bottom));
002730
002731          IF rErase IN rActions THEN
```

```
002732              SELF.Erase;
002733
002734          IF (rDraw IN rActions) OR (highTransit <> hNone) OR needGray THEN
002735              BEGIN
002736              PushFocus;
002737              SELF.Focus;
002738
002739              IF needGray THEN
002740                  BEGIN
002741                  PenNormal;
002742                  PenSize(2, 2);
002743
002744                  {draw the vertical strip of gray ...}
002745                  tempLRect := viewedLRect;
002746                  tempLRect.left := viewExtentLRect.right;
002747                  FillLRect(tempLRect, lPatLtGray);
002748
002749                  {... then the horizontal strip ...}
002750                  tempLRect := viewedLRect;
002751                  tempLRect.top := viewExtentLRect.bottom;
002752                  FillLRect(tempLRect, lPatLtGray);
002753
002754                  {... then frame the bottom right of the view extent with a 2-pixel line outside the extent;
002755                      note that the topLeft does not matter}
002756                  tempLRect.topLeft := viewedLRect.topLeft;
002757                  tempLRect.botRight := viewExtentLRect.botRight;
002758
002759                  InsetLRect(tempLRect, -2, -2);
002760                  FrameLRect(tempLRect);
002761                  END;
002762
002763              IF rDraw IN rActions THEN
002764                  SELF.currentView.Draw;
002765
002766              IF highTransit <> hNone THEN
002767                  IF panel.previewMode = mPrvwMargins THEN
002768                      panel.paginatedView.DoOnPages(TRUE, HighlightOnThePad)
002769                  ELSE
002770                      HighlightOnThePad;
002771
002772              IF rDraw IN rActions THEN  {Page breaks after highlighting, in case highlighting doesn't XOR}
002773                  IF panel.previewMode = mPrvwBreaks THEN  {Xors automatic as well as manual page breaks}
002774                      SELF.currentView.printManager.DrawBreaks(FALSE);
002775
002776              PopFocus;
002777              END;
002778          {$IFC fTrace}EP;{$ENDC}
002779      END;
```

```
002780  {$S SgABCres}
002781
002782
002783      {$S sRes}
002784      PROCEDURE {TPane.}Resize{(newInnerRect: Rect; vhs: VHSelect)};
002785          VAR innerRect:      Rect;
002786              paneLongSize:   LPoint;
002787      BEGIN
002788          {$IFC fTrace}BP(7);{$ENDC}
002789          innerRect := SELF.innerRect;
002790          AlignRect(innerRect, newInnerRect, vhs);
002791          SELF.SetInnerRect(innerRect);
002792          SELF.clippedRect := innerRect;
002793          SELF.DistToLDist(Point(FDiagRect(innerRect)), paneLongSize);
002794    {$H-} LPtPlusLPt(SELF.viewedLRect.topLeft, paneLongSize, SELF.viewedLRect.botRight); {$H+}
002795          SELF.availLRect := SELF.viewedLRect;
002796    {$H-} InsetLRect(SELF.availLRect, -8192, -8192); {$H+}
002797          {$IFC fTrace}EP;{$ENDC}
002798      END;
002799
002800
002801      {$S sScroll}
002802      PROCEDURE {TPane.}ScrollBy(VAR deltaLPt: LPoint);
002803          VAR panel:      TPanel;
002804              deltaPt:    Point;
002805              vhs:        VHSelect;
002806              band:       TBand;
002807              tempPt:     Point;
002808      BEGIN
002809          {$IFC fTrace}BP(5);{$ENDC}
002810          panel := SELF.panel;
002811
002812          IF panel.panes.Size = 1 THEN
002813              BEGIN
002814              PushFocus;
002815              SELF.panel.window.Focus;
002816
002817              panel.DoScrolling(SELF, SELF, TRUE, TRUE, deltaLPt);
002818              IF NOT EqualLPt(deltaLPt, zeroLPt) THEN
002819                  BEGIN
002820                  SELF.OffsetBy(deltaLPt);
002821                  FOR vhs := v TO h DO
002822                      panel.scrollBars[vhs].firstBox.MoveThumb(TBand(panel.bands[vhs].First).ThumbPos);
002823                  END;
002824
002825              PopFocus;
002826              END
002827          ELSE
```

```
002828              FOR vhs := v TO h DO
002829                  BEGIN
002830                  band := TBand(panel.ChildWithPt(SELF.innerRect.topLeft, panel.bands[vhs], tempPt));
002831                  band.ScrollBy(deltaLPt.vh[vhs]);
002832                  END;
002833          {$IFC fTrace}EP;{$ENDC}
002834      END;
002835
002836
002837      {$S sScroll}
002838      PROCEDURE {TPane.}ScrollToReveal(VAR anLRect: LRect; hMinToSee, vMinToSee: INTEGER);
002839          VAR ptMinToSee:     Point;
002840              minToSee:       INTEGER;
002841              viewedLRect:    LRect;
002842              deltaLPt:       LPoint;
002843              vhs:            VHSelect;
002844              lcd:            LONGINT;
002845      BEGIN
002846          {$IFC fTrace}BP(5);{$ENDC}
002847          viewedLRect := SELF.viewedLRect;
002848          SetPt(ptMinToSee, hMinToSee, vMinToSee);
002849
002850          FOR vhs := v TO h DO
002851              BEGIN
002852              minToSee := Min(LengthRect(SELF.innerRect, vhs), ptMinToSee.vh[vhs]);
002853
002854              lcd := anLRect.topLeft.vh[vhs] + minToSee - viewedLRect.botRight.vh[vhs];
002855              IF lcd <= 0 THEN
002856                  BEGIN
002857                  lcd := anLRect.botRight.vh[vhs] - minToSee - viewedLRect.topLeft.vh[vhs];
002858                  IF lcd >= 0 THEN
002859                      lcd := 0;
002860                  END;
002861              deltaLPt.vh[vhs] := lcd;
002862              END;
002863
002864          SELF.ScrollBy(deltaLPt);
002865          {$IFC fTrace}EP;{$ENDC}
002866      END;
002867
002868
002869      {$S SgABCcld}
002870      PROCEDURE {TPane.}SetZoomFactor{(zoomNumerator, zoomDenominator: Point)};
002871          VAR zoomFactor: TScaler;
002872              newLRight:   LONGINT;
002873              newLBottom:  LONGINT;
002874              newViewedLRect: LRect;
002875      BEGIN
```

```
002876          {$IFC fTrace}BP(9);{$ENDC}
002877          Reduce(zoomNumerator.h, zoomDenominator.h); {reduce to lowest terms}
002878          Reduce(zoomNumerator.v, zoomDenominator.v);
002879
002880         {adjust viewed lRect}
002881          newLRight := Min(
002882              (SELF.viewedLRect.right * zoomDenominator.h * SELF.zoomFactor.numerator.h)
002883                  DIV ( zoomNumerator.h * SELF.zoomFactor.denominator.h),
002884               SELF.currentView.extentLRect.right);
002885          newLBottom := Min(
002886              (SELF.viewedLRect.bottom * zoomDenominator.v * SELF.zoomFactor.numerator.v)
002887                 DIV ( zoomNumerator.v * SELF.zoomFactor.denominator.v),
002888               SELF.currentView.extentLRect.bottom);
002889
002890          SetLRect(newViewedLRect, SELF.viewedLRect.left, SELF.viewedLRect.top,
002891                    newLRight, newLBottom);
002892          SetPt(zoomFactor.numerator, zoomNumerator.h, zoomNumerator.v);
002893          SetPt(zoomFactor.denominator, zoomDenominator.h, zoomDenominator.v);
002894          SELF.Redefine(SELF.innerRect, newViewedLRect, SELF.padRes, SELF.viewedRes, zoomFactor, SELF.port);
002895          {$IFC fTrace}EP;{$ENDC}
002896      END;
002897      {$S SgABCres}
002898
002899
002900  {$S SgABCini}
002901  END;
002902  {$S SgABCres}
002903
002904
002905
002906  METHODS OF TMarginPad;
002907
002908
002909      {$S SgABCini}
002910      FUNCTION {TMarginPad.}CREATE{(object: TObject; heap: THeap): TMarginPad};
002911          VAR bodyPad: TBodyPad;
002912      BEGIN
002913          {$IFC fTrace}BP(9);{$ENDC}
002914          IF object = NIL THEN
002915              object := NewObject(heap, THISCLASS);
002916          SELF := TMarginPad(object);
002917
002918          bodyPad := TBodyPad.CREATE(NIL, heap, SELF);
002919          SELF.bodyPad := bodyPad;
002920          {$IFC fTrace}EP;{$ENDC}
002921      END;
002922      {$S SgABCres}
002923
```

```
002924
002925      {$S SgABCpri}
002926      PROCEDURE {TMarginPad.}Rework{(itsView: TView; itsOrigin: Point; itsRes: Point;
002927                        itsPageNumber: LONGINT; itsZoomFactor: TScaler; itsPort: GrafPtr)};
002928          VAR itsViewedLRect:  LRect;
002929              printerMetrics:  TPrinterMetrics;
002930              bodyPad:         TBodyPad;
002931              innerRect:       Rect;
002932
002933          PROCEDURE ScaleToPadSpace(printRect: Rect; VAR padRect: Rect);
002934              VAR padLRect: LRect;
002935          {NB: itsOrigin is a free var in this proc}
002936          BEGIN
002937          SetLRect(padLRect,
002938              LIntOvrInt(ORD4(printRect.left)   * itsRes.h * itsZoomFactor.numerator.h,   {this whole stmt}
002939                         printerMetrics.res.h * itsZoomFactor.denominator.h),
002940              LIntOvrInt(ORD4(printRect.top)    * itsRes.v * itsZoomFactor.numerator.v,
002941                         printerMetrics.res.v * itsZoomFactor.denominator.v),
002942              LIntOvrInt(ORD4(printRect.right)  * itsRes.h * itsZoomFactor.numerator.h,
002943                         printerMetrics.res.h * itsZoomFactor.denominator.h),
002944              LIntOvrInt(ORD4(printRect.bottom) * itsRes.v * itsZoomFactor.numerator.v,
002945                         printerMetrics.res.v * itsZoomFactor.denominator.v));
002946          noPad.LRectToRect(padLRect, padRect);
002947          OffsetRect(padRect, itsOrigin.h, itsOrigin.v);
002948          END;
002949
002950      BEGIN
002951          {$IFC fTrace}BP(7);{$ENDC}
002952          SELF.view := itsView;
002953          printerMetrics := SELF.view.printManager.printerMetrics;
002954          SELF.pageNumber := itsPageNumber;
002955
002956          ScaleToPadSpace(printerMetrics.paperRect, innerRect);
002957
002958          SELF.Redefine(innerRect, SELF.view.printManager.paperLRect,
002959                                itsRes,           {pad resolutions}
002960                                itsView.res,      {viewed resolutions}
002961                                itsZoomFactor, itsPort);        {calls TPad's Redefine method}
002962        {page's 'viewed space' has same metrics as the owning view's}
002963
002964          SELF.bodyPad.Recompute;
002965          {$IFC fTrace}EP;{$ENDC}
002966      END;
002967      {$S SgABCres}
002968
002969
002970      {$S SgABCini}
002971      PROCEDURE {TMarginPad.}Free;
```

```
002972      BEGIN
002973          {$IFC fTrace}BP(6);{$ENDC}
002974          Free(SELF.bodyPad);
002975          TObject.Free;
002976          {$IFC fTrace}EP;{$ENDC}
002977      END;
002978      {$S SgABCres}
002979
002980
002981      {$S SgABCpri}
002982      PROCEDURE {TMarginPad.}SetForPage{(itsPageNumber: LONGINT; itsOrigin: Point)};
002983          VAR innerRect:  Rect;
002984              newOffset:  LPoint;
002985      BEGIN
002986          {$IFC fTrace}BP(7);{$ENDC}
002987          SELF.pageNumber := itsPageNumber;
002988          innerRect := SELF.innerRect;
002989          OffsetRect(innerRect, itsOrigin.h - SELF.innerRect.left, itsOrigin.v - SELF.innerRect.top);
002990          SELF.SetInnerRect(innerRect);
002991          SELF.clippedRect := innerRect;
002992          WITH innerRect DO
002993              SetLPt(newOffset, - left, - top);
002994          SELF.SetScrollOffset(newOffset);
002995          SELF.bodyPad.SetForPage(itsPageNumber);
002996          {$IFC fTrace}EP;{$ENDC}
002997      END;
002998      {$S SgABCres}
002999
003000
003001      {$IFC fDebugMethods}
003002      {$S SgABCdbg}
003003      PROCEDURE {TMarginPad.}Fields{(PROCEDURE Field(nameAndType: S255))};
003004      BEGIN
003005          TPad.Fields(Field);
003006          Field('view: TView');
003007          Field('pageNumber: LONGINT');
003008          Field('bodyPad: TBodyPad');
003009          Field('');
003010      END;
003011      {$S SgABCres}
003012      {$ENDC}
003013
003014
003015      {$IFC fDbgABC}
003016      {$S SgABCdbg}
003017      FUNCTION TMarginPad.BindHeap{(activeVsClip, doBind: BOOLEAN): THeap}; {called by HeapDump in UOBJECT2}
003018      BEGIN
003019          {$IFC fMaxTrace}BP(1);{$ENDC}
```

```
003020          {$IFC fMaxTrace}EP;{$ENDC}
003021          BindHeap := NIL;
003022      (*  IF activeWindowID <> 0 THEN {don't allow inactive windows to use this} -- WHY NOT????  *)
003023              BEGIN
003024              IF activeVsClip THEN
003025                  BEGIN
003026      (*          IF (currentDocument <> NIL) AND doBind THEN
003027                      BindHeap := currentDocument.docHeap;                                    *)
003028                  IF (boundDocument <> NIL) AND doBind THEN
003029                      BindHeap := boundDocument.docHeap;
003030                  END
003031              ELSE
003032              IF currentDocument <> clipboard THEN
003033                  IF doBind THEN
003034                      BEGIN
003035                      hadToBindClip := boundClipboard = NIL;
003036                      IF hadToBindClip THEN
003037                          clipboard.Bind;
003038                      BindHeap := clipboard.docHeap;
003039                      END
003040                  ELSE IF hadToBindClip THEN
003041                      clipboard.Unbind;
003042              END;
003043      END;
003044      {$S SgABCres}
003045      {$ENDC}
003046
003047
003048      {$S SgABCcld}
003049      PROCEDURE {TMarginPad.}Crash;
003050      BEGIN   {SELF = crashPad, presumably, but in any case, someone wants this process to die, so...}
003051          IF isInitialized THEN
003052              process.Complete(FALSE);
003053      END;
003054      {$S SgABCres}
003055
003056
003057      PROCEDURE TMarginPad.SetScrollOffset(VAR newOffset: LPoint);  {+SW+}
003058      BEGIN
003059          {$IFC fTrace}BP(7);{$ENDC}
003060          IF fExperimenting OR NOT amPrinting THEN
003061              SUPERSELF.SetScrollOffset(newOffset)
003062          ELSE
003063              WITH SELF DO
003064                  BEGIN
003065                  scrollOffset := newOffset;
003066                  origin := zeroPt;
003067                  cdOffset := newOffset;
```

```
003068                END;
003069            {$IFC fTrace}EP;{$ENDC}
003070        END;
003071
003072
003073    {$S SgABCini}
003074    END;
003075    {$S SgABCres}
003076
003077
003078    METHODS OF TBodyPad;
003079
003080
003081        {$S SgABCini}
003082        FUNCTION {TBodyPad.}CREATE{(object: TObject; heap: THeap; itsMarginPad: TMarginPad): TBodyPad};
003083        BEGIN
003084            {$IFC fTrace}BP(9);{$ENDC}
003085            IF object = NIL THEN
003086                object := NewObject(heap, THISCLASS);
003087            SELF := TBodyPad(object);
003088
003089            SELF.marginPad := itsMarginPad;
003090            {$IFC fTrace}EP;{$ENDC}
003091        END;
003092        {$S SgABCres}
003093
003094
003095        {$IFC fDebugMethods}
003096        {$S SgABCdbg}
003097        PROCEDURE {TBodyPad.}Fields{(PROCEDURE Field(nameAndType: S255))};
003098        BEGIN
003099            TPad.Fields(Field);
003100            Field('marginPad: TMarginPad');
003101            Field('nonNullBody: Rect');
003102        END;
003103        {$S SgABCres}
003104        {$ENDC}
003105
003106
003107        {$S SgABCpri}
003108        PROCEDURE {TBodyPad.}Focus;
003109        BEGIN
003110            {$IFC fTrace}BP(6);{$ENDC}
003111            SELF.ClipFurtherTo(SELF.nonNullBody);
003112            TPad.Focus;
003113           {$IFC fTrace}EP;{$ENDC}
003114        END;
003115        {$S SgABCres}
```

```
003116
003117
003118      {$S SgABCpri}
003119      PROCEDURE {TBodyPad.}Recompute;
003120          VAR myViewedLRect:  LRect;
003121              myInnerRect:    Rect;
003122              view:           TView;
003123              marginPad:      TMarginPad;
003124              bodyRect:       Rect;
003125              printManager:   TPrintManager;
003126      BEGIN
003127          {$IFC fTrace}BP(7);{$ENDC}
003128          marginPad := SELF.marginPad;
003129          view := marginPad.view;
003130          printManager := view.printManager;
003131          printManager.GetPageLimits(marginPad.pageNumber, myViewedLRect);
003132
003133          marginPad.LRectToRect(printManager.contentLRect, myInnerRect);
003134          WITH marginPad.origin DO {$H-}
003135              OffsetRect(myInnerRect, -h, -v); {$H+}
003136
003137          SELF.Redefine(myInnerRect, myViewedLRect, marginPad.padRes,
003138            view.res, marginPad.zoomFactor, SELF.marginPad.port);
003139          bodyRect.topLeft := SELF.innerRect.topLeft;
003140          SELF.LPtToPt(myViewedLRect.botRight, bodyRect.botRight);
003141          bodyRect.botRight := Point(FPtMinusPt(bodyRect.botRight, SELF.origin));
003142
003143          SELF.nonNullBody := bodyRect;
003144          {$IFC fTrace}EP;{$ENDC}
003145      END;
003146      {$S SgABCres}
003147
003148
003149      {$S SgABCpri}
003150      PROCEDURE {TBodyPad.}SetForPage{(itsPageNumber: LONGINT)};
003151          VAR myViewedLRect:  LRect;
003152              myInnerRect:    Rect;
003153              bodyRect:       Rect;
003154              printManager:   TPrintManager;
003155              newOffset:      LPoint;
003156      BEGIN
003157          {$IFC fTrace}BP(7);{$ENDC}
003158          printManager := SELF.marginPad.view.printManager;
003159          printManager.GetPageLimits(itsPageNumber, myViewedLRect);
003160          SELF.marginPad.LRectToRect(printManager.contentLRect, myInnerRect);
003161          WITH SELF.marginPad.origin DO {$H-}
003162              OffsetRect(myInnerRect, -h, -v); {$H+}
003163
```

```
003164          SELF.SetInnerRect(myInnerRect);
003165
003166          WITH SELF, newOffset, scaleFactor DO
003167              BEGIN
003168              viewedLRect := myViewedLRect;
003169              availLRect := myViewedLRect;
003170        {$H-} InsetLRect(availLRect, -8192, -8192);  {$H+}
003171              clippedRect := myInnerRect;
003172              IF scaled THEN
003173                  BEGIN
003174        {$H-}     h := LIntOvrInt(LIntMulInt(myViewedLRect.left, numerator.h), denominator.h) {+++LSR+++}
003175                           - myInnerRect.left;
003176                  v := LIntOvrInt(LIntMulInt(myViewedLRect.top, numerator.v), denominator.v) {+++LSR+++}
003177                          - myInnerRect.top;  {$H+}
003178                  END
003179              ELSE
003180                  BEGIN
003181                  h := myViewedLRect.left - myInnerRect.left;
003182                  v := myViewedLRect.top - myInnerRect.top;
003183                  END;
003184              END;
003185          SELF.SetScrollOffset(newOffset);
003186
003187          SELF.nonNullBody := SELF.innerRect;
003188   {$H-}  SELF.LPtToPt(myViewedLRect.botRight, SELF.nonNullBody.botRight);
003189          SELF.nonNullBody.botRight := Point(FPtMinusPt(SELF.nonNullBody.botRight, SELF.origin)); {$H+}
003190          {$IFC fTrace}EP;{$ENDC}
003191      END;
003192      {$S SgABCres}
003193
003194
003195      PROCEDURE TBodyPad.SetScrollOffset(VAR newOffset: LPoint);  {+SW+}
003196      BEGIN
003197          {$IFC fTrace}BP(7);{$ENDC}
003198          IF fExperimenting OR NOT amPrinting THEN
003199              SUPERSELF.SetScrollOffset(newOffset)
003200          ELSE
003201              WITH SELF DO
003202                  BEGIN
003203                  scrollOffset := newOffset;
003204                  origin := zeroPt;
003205                  cdOffset := newOffset;
003206                  END;
003207          {$IFC fTrace}EP;{$ENDC}
003208      END;
003209
003210
003211  {$S SgABCini}
```

```
003212  END;
003213  {$S SgABCres}
003214
003215
003216  METHODS OF TScroller;
003217
003218
003219      {$S sCldInit}
003220      FUNCTION  {TScroller.}CREATE{(object: TObject; heap: THeap; itsScrollBar: TScrollBar; itsId: TSBoxID)
003221                                   :TScroller};
003222      BEGIN
003223          {$IFC fTrace}BP(7);{$ENDC}
003224          IF object = NIL THEN
003225              object := NewObject(heap, THISCLASS);
003226          SELF := TScroller(object);
003227
003228          WITH SELF DO
003229              BEGIN
003230              scrollBar := itsScrollBar;
003231              band := NIL;
003232              sBoxID := itsId;
003233        {$H-} SetSbRefcon(POINTER(sBoxID), ORD(SELF)); {$H+}
003234              END;
003235          {$IFC fTrace}EP;{$ENDC}
003236      END;
003237      {$S SgABCres}
003238
003239
003240      {$S SgABCini}
003241      PROCEDURE {TScroller.}Free;
003242          VAR sbList: TSbList;
003243      BEGIN
003244          {$IFC fTrace}BP(7);{$ENDC}
003245          PreSbList(sbList, SELF.scrollBar);
003246    {$H-} KillSb(sbList, POINTER(SELF.sBoxID)); {$H+}
003247          PostSbList(sbList, SELF.scrollBar);
003248          TObject.Free;
003249          {$IFC fTrace}EP;{$ENDC}
003250      END;
003251      {$S SgABCres}
003252
003253
003254      {$IFC fDebugMethods}
003255      {$S SgABCdbg}
003256      PROCEDURE {TScroller.}Fields{(PROCEDURE Field(nameAndType: S255))};
003257      BEGIN
003258         Field('scrollBar: TScrollBar');
003259         Field('band: TBand');
```

```
003260          Field('sBoxID: LONGINT');
003261      END;
003262      {$S SgABCres}
003263      {$ENDC}
003264
003265
003266      {$S sScroll}
003267      PROCEDURE {TScroller.}FillIcon{(icon: TEnumIcons; fBlack: BOOLEAN)};
003268          TYPE TIconAlias =
003269                  RECORD
003270                  CASE INTEGER OF
003271                      1: (sblib: TIcon);
003272                      2: (abc:   TEnumIcons);
003273                  END;
003274          VAR iconAlias:   TIconAlias;
003275      BEGIN
003276          {$IFC fMaxTrace}BP(1);{$ENDC}
003277          {$IFC fMaxTrace}EP;{$ENDC}
003278          iconAlias.abc := icon;
003279          PaintArw(POINTER(SELF.sBoxID), iconAlias.sblib, fBlack);
003280      END;
003281
003282
003283      {$S sRes}
003284      PROCEDURE {TScroller.}GetSize{(VAR boxRect: Rect)};
003285      BEGIN
003286          {$IFC fTrace}BP(3);{$ENDC}
003287          GetSbRect(POINTER(SELF.sBoxID), boxRect);
003288          {$IFC fTrace}EP;{$ENDC}
003289      END;
003290
003291
003292      {$S sScroll}
003293      PROCEDURE {TScroller.}MoveThumb{(newThumbPos: INTEGER)};
003294          {NOTE: assumes we are focused on the window, NOT on a pane}
003295      BEGIN
003296          {$IFC fTrace}BP(4);{$ENDC}
003297          IF activeWindowID <> 0 THEN
003298              BEGIN
003299              SetupMvThumb(POINTER(SELF.sboxID));
003300              MoveThumb(newThumbPos);
003301              END;
003302          {$IFC fTrace}EP;{$ENDC}
003303      END;
003304
003305
003306      {$S sSplit}
003307      PROCEDURE {TScroller.}ResplitAt{(newSkwrCd: INTEGER; prevScroller: TScroller)};
```

```
003308          VAR vhs:        VHSelect;
003309              sbRect:     Rect;
003310              prevSbRect: Rect;
003311              hsb:        THSb;
003312              deltaCd:    INTEGER;
003313              minSize:    INTEGER;
003314      BEGIN
003315          {$IFC fTrace}BP(7);{$ENDC}
003316          vhs := SELF.ScrollDir;
003317          minSize := dptSkewer.vh[vhs];
003318          hsb := POINTER(SELF.sBoxID);
003319          GetSbRect(hsb, sbRect);
003320          prevScroller.GetSize(prevSbRect);
003321
003322          {If either scroller to becomes too small, delete it}
003323          IF newSkwrCd <= prevSbRect.topLeft.vh[vhs] + minSize THEN
003324              newSkwrCd := prevSbRect.topLeft.vh[vhs]
003325          ELSE IF newSkwrCd >= sbRect.botRight.vh[vhs] - minSize THEN
003326              newSkwrCd := sbRect.botRight.vh[vhs];
003327
003328          deltaCd := newSkwrCd - sbRect.topLeft.vh[vhs];
003329          AdjSplitBetween(POINTER(prevScroller.sBoxID), hsb, deltaCd);
003330          {$IFC fTrace}EP;{$ENDC}
003331      END;
003332      {$S SgABCres}
003333
003334
003335      {$S sRes}
003336      FUNCTION  {TScroller.}ScrollDir{: VHSelect};
003337      BEGIN
003338          {$IFC fTrace}BP(3);{$ENDC}
003339          ScrollDir := TyVHOfSb(POINTER(SELF.sBoxID));
003340          {$IFC fTrace}EP;{$ENDC}
003341      END;
003342
003343
003344      {$S sRes}
003345      PROCEDURE {TScroller.}SetSize{(ownerRect: Rect)};
003346          VAR sbRect: Rect;
003347              vhs: VHSelect;
003348              width:  INTEGER;
003349
003350                  {ownerRect is the band's outerRect.
003351                   For v bar: top/bottom  = ownerRect top/bottom
003352                              left        = ownerRect right - 1
003353                              right       = left + dhSBox}
003354      BEGIN
003355          {$IFC fTrace}BP(7);{$ENDC}
```

```
003356          vhs := orthogonal[SELF.ScrollDir];
003357          sbRect := ownerRect;
003358          sbRect.topLeft.vh[vhs] := sbRect.botRight.vh[vhs] - 1;
003359          IF SELF.scrollBar.isVisible THEN
003360              width := dptSbox.vh[vhs]
003361          ELSE
003362              width := 0;
003363          sbRect.botRight.vh[vhs] := sbRect.topLeft.vh[vhs] + width;
003364
003365          SetSbRect(POINTER(SELF.sBoxID), sbRect);
003366          {$IFC fTrace}EP;{$ENDC}
003367      END;
003368      {$S SgABCres}
003369
003370
003371      {$S sSplit}
003372      PROCEDURE {TScroller.}SplitAt{(newSkwrCd: INTEGER; VAR nextScroller: TScroller)};
003373          VAR newHsb: THsb;
003374              sbList: TSbList;
003375      BEGIN
003376          {$IFC fTrace}BP(7);{$ENDC}
003377          PreSbList(sbList, SELF.scrollBar);
003378          SplitSb(sbList, POINTER(SELF.sBoxID), newHsb, newSkwrCd);
003379          PostSbList(sbList, SELF.scrollBar);
003380          nextScroller := TScroller.CREATE(NIL, SELF.Heap, SELF.scrollBar, ORD(newHsb));
003381          {$IFC fTrace}EP;{$ENDC}
003382      END;
003383      {$S SgABCres}
003384
003385
003386      {$S sScroll}
003387      FUNCTION  {TScroller.}ThumbRange{: INTEGER};
003388          VAR posts: TPosts;
003389      BEGIN
003390          {$IFC fTrace}BP(7);{$ENDC}
003391          MkPosts(POINTER(SELF.sBoxID), posts);
003392          ThumbRange := posts[iconGryB] - posts[iconPagA] - dptThumb.vh[SELF.ScrollDir];
003393          {$IFC fTrace}EP;{$ENDC}
003394      END;
003395
003396
003397      {$S sSplit}
003398      PROCEDURE {TScroller.}TrackSkewer{(mousePt: Point; VAR newSkwrCd: INTEGER;
003399                                         VAR scroller, prevScroller: TScroller)};
003400          VAR hsb, prevHsb:   THsb;
003401              sbList:         TSbList;
003402              limitRect:      Rect;
003403              newSkwrPt:      Point;
```

```
003404      BEGIN
003405          {$IFC fTrace}BP(7);{$ENDC}
003406          hsb := POINTER(SELF.sBoxID);
003407          FixRLimits(hsb, limitRect);
003408          AlignRect(limitRect, SELF.band.outerRect, orthogonal[SELF.ScrollDir]);
003409          DragSkewer(hsb, mousePt, limitRect, newSkwrPt);
003410          newSkwrCd := newSkwrPt.vh[SELF.ScrollDir];
003411          prevHsb:= HsbPrev(hsb);
003412          IF prevHsb = hsbNil THEN
003413              BEGIN
003414              PreSbList(sbList, SELF.scrollBar);
003415              hsb := HsbFromPt(sbList, newSkwrPt);
003416              PostSbList(sbList, SELF.scrollBar);
003417              IF hsb = hsbNil THEN
003418                  scroller := NIL
003419              ELSE
003420                  scroller := POINTER(RefconSb(hsb));
003421              prevScroller := NIL;
003422              END
003423          ELSE
003424              BEGIN
003425              scroller := SELF;
003426              prevScroller := POINTER(RefconSb(prevHsb));
003427              END;
003428          {$IFC fTrace}EP;{$ENDC}
003429      END;
003430
003431
003432      {$S sScroll}
003433      PROCEDURE {TScroller.}TrackThumb{(mousePt: Point; VAR oldThumbPos, newThumbPos: INTEGER)};
003434      BEGIN
003435          {$IFC fTrace}BP(7);{$ENDC}
003436          oldThumbPos := CThumbPos(POINTER(SELF.sBoxID));
003437          DragThumb(POINTER(SELF.sBoxID), mousePt, newThumbPos);
003438          {$IFC fTrace}EP;{$ENDC}
003439      END;
003440
003441
003442  {$S SgABCini}
003443  END;
003444  {$S SgABCres}
003445
003446
003447  METHODS OF TScrollBar;
003448
003449
003450      {$S SgABCini}
003451      FUNCTION  {TScrollBar.}CREATE{(object: TObject; heap: THeap; vhs: VHSelect; outerRect: Rect;
```

```
003452                                  itsVisibility: BOOLEAN): TScrollBar};
003453          VAR sbList:     TSbList;
003454              hsb:        THSb;
003455              firstBox:   TScroller;
003456      BEGIN
003457          {$IFC fTrace}BP(7);{$ENDC}
003458          IF object = NIL THEN
003459              object := NewObject(heap, THISCLASS);
003460          SELF := TScrollBar(object);
003461
003462          InitSbList(sbList, POINTER(ORD(heap)));
003463          hsb := SbCreate(sbList, hsbNil, vhs, zeroPt, 0);
003464          PostSbList(sbList, SELF);
003465
003466          firstBox := TScroller.CREATE(NIL, heap, SELF, ORD(hsb));
003467          SELF.firstBox := firstBox;
003468
003469          SELF.ChangeVisibility(itsVisibility, outerRect, []);       {The band's outerRect}
003470          {$IFC fTrace}EP;{$ENDC}
003471      END;
003472      {$S SgABCres}
003473
003474
003475      {$IFC fDebugMethods}
003476      {$S SgABCdbg}
003477      PROCEDURE {TScrollBar.}Fields{(PROCEDURE Field(nameAndType: S255))};
003478      BEGIN
003479          Field('firstBox: TScroller');
003480          Field('isVisible: BOOLEAN');
003481          Field('');
003482      END;
003483      {$S SgABCres}
003484      {$ENDC}
003485
003486
003487      {$S sCldInit}
003488      PROCEDURE {TScrollBar.}ChangeVisibility{(needsBothBars: BOOLEAN;
003489                                               bandOuterRect: Rect; itsAbilities: TAbilities)};
003490          VAR hsb:            THsb;
003491              scroller:       TScroller;
003492              needsThisBar:   BOOLEAN;
003493              icons:          TSIcon;
003494      BEGIN
003495          {$IFC fTrace}BP(7);{$ENDC}
003496          needsThisBar := needsBothBars OR (aBar IN itsAbilities);
003497          SELF.isVisible := needsThisBar;
003498          icons := [];
003499          IF needsThisBar THEN
```

```
003500              BEGIN {if no bar, then no icons}
003501              IF aScroll IN itsAbilities THEN
003502                  icons := icons + [iconArwA, iconArwB, iconThumb, iconPagA, iconPagB];
003503              IF aSplit IN itsAbilities THEN
003504                  icons := icons + [iconSkewer];
003505              END;
003506          hsb := POINTER(SELF.firstBox.sBoxID);
003507          WHILE hsb <> hsbNil DO
003508              BEGIN
003509              scroller := TScroller(RefconSb(hsb));
003510              IF scroller.band <> NIL THEN
003511                  bandOuterRect := scroller.band.outerRect;
003512              scroller.SetSize(bandOuterRect);
003513              SetSbIcons(hsb, icons);
003514              hsb := HsbNext(hsb);
003515              END;
003516          {$IFC fTrace}EP;{$ENDC}
003517      END;
003518
003519
003520      {$S sScroll}
003521      FUNCTION  {TScrollBar.}DownAt{(mousePt: Point; VAR scroller: TScroller; VAR icon: TEnumIcons): BOOLEAN};
003522          TYPE TIconAlias =
003523                  RECORD
003524                  CASE INTEGER OF
003525                      1: (sblib: TIcon);
003526                      2: (abc:   TEnumIcons);
003527                  END;
003528          VAR iconAlias:   TIconAlias;
003529              hsbHit:      THSb;
003530              sbList:      TSbList;
003531      BEGIN
003532          {$IFC fTrace}BP(7);{$ENDC}
003533          DownAt := FALSE;
003534
003535          IF SELF.isVisible THEN
003536              BEGIN
003537              PreSbList(sbList, SELF);
003538        {$H-} IF FSbHit(sbList, mousePt, hsbHit, iconAlias.sbLib) {$H+} THEN
003539                  BEGIN
003540                  DownAt := TRUE;
003541                  scroller := POINTER(RefconSb(hsbHit));
003542                  icon := iconAlias.abc;
003543                  END;
003544              END;
003545          {$IFC fTrace}EP;{$ENDC}
003546      END;
003547
```

```
003548
003549      {$S sStartup}
003550      PROCEDURE {TScrollBar.}Draw;
003551      BEGIN
003552          {$IFC fTrace}BP(7);{$ENDC}
003553          IF SELF.isVisible THEN
003554              PaintSbar(POINTER(SELF.firstBox.sBoxID));
003555          {$IFC fTrace}EP;{$ENDC}
003556      END;
003557
003558
003559      {$S sStartup}
003560      PROCEDURE {TScrollBar.}Erase;
003561      BEGIN
003562          {$IFC fTrace}BP(7);{$ENDC}
003563          IF SELF.isVisible THEN
003564              EraseSBar(POINTER(SELF.firstBox.sBoxID));
003565          {$IFC fTrace}EP;{$ENDC}
003566      END;
003567
003568
003569  {$S SgABCini}
003570  END;
003571  {$S SgABCres}
003572  {$S SgABCini}
003573

End of File -- Lines: 3573 Characters: 114796
```

```
================================================================================================
FILE: "LIBTK/UDIALOG.TEXT"
================================================================================================

000001  (*        >>>>>>>>  U D I A L O G  <<<<<<<<
000002
000003  *)
000004
000005  {$SETC forOS := TRUE}
000006
000007  UNIT UDialog;  {Copyright 1984 by Apple Computer, Inc}
000008
000009  {04/25/84 0015  Added field TEditLegendSelection.tripleClick, and methods TEditLegendSelection.
000010                       MousePress, MouseMove, and MouseRelease}
000011  {04/23/84 1210  Removed all references to 'underEdit' field of TDialogImage}
000012
000013  {$Setc IsIntrinsic := TRUE }
000014
000015  {$IFC IsIntrinsic}
000016  INTRINSIC;
000017  {$ENDC}
000018
000019
000020  INTERFACE
000021
000022  USES
000023    {$U libtk/UObject}          UObject,
000024  {$IFC LibraryVersion <= 20}
000025    {$U UFont}            UFont,
000026  {$ENDC}
000027    {$U QuickDraw}        QuickDraw,
000028    {$U libtk/UDraw}            UDraw,
000029
000030    {$U libtk/UABC}             UABC,
000031    {$U libtk/UUnivText}        UTKUniversalText,
000032    {$U libtk/UText}            UText;
000033
000034
000035  CONST
000036      UDialogVersion  = 'UDialog 25Apr84 16:30';
000037
000038  (*
000039                            -----  Dialog Building Block for the ToolKit ------
000040
000041
000042
000043   The Dialog Building Block provides the following standard kinds of dialog Images:
```

```
000044
000045         Button           A Lisa-style button (a round-cornered Rectangle for pushing, with text inside it)
000046         Checkbox         A checkbox (a box for checking, plus an optional associated textual label)
000047         Cluster          A set of related checkboxes of which only one is selected at a time
000048         InputFrame       A place for keyboard input to be inhaled
000049         Legend           A character string, together with font & face information
000050
000051         TextDialogImage  A box of text managed by the Text editor  (largely untested)
000052         PicObject        A QuickDraw picture  (never tested; probably not bankable; status uncertain)
000053
000054    The basic bankable dialog entity which can be stashed into/retrieved from a Resource File
000055    is the class TDialog.  For each different kind of dialog box you want, you will typically define
000056    another subclass of TDialog.
000057
000058    To EDIT a dialog interactively, you must:
000059         (1)  Have the menu items '´Edit Dialog' and 'Stop Editing Dialog' in your phrase-file
000060         (2)  If the dialog is viewed in your main window rather than in a dialog box, (such as Preferences)
000061                then your own main Window.CanDoCmd should enable uEditDialog whenever the dialog to be editted
000062                is unambiguously selected in the window and there is not a dialog box up; in this
000063                case, the dialog editting takes place in a dialog box whereas the dialog itself resides
000064                in the main window.
000065
000066         CAUTION: Until Resource Files are incoporated, the edits to a dialog are local to the document
000067                  in which you made the edits, as well as documents made from a stationery pad made from
000068                  that document.
000069
000070      How to have your own view be a subclass of TDialogView, and still do all of its normal View things,
000071      while having the Dialog Building Block handle everything that occurs which is relevant to
000072      its dialogs:
000073
000074       (a) To draw the non-dialog parts of the view, implement method TDialogView.XDraw
000075       (b) To set the cursor in the non-dialog parts of the view, implement method TDialogView.XCursorAt
000076       (c) Implement XMousePress, XMouseMove, and XMouseRelease instead of their non-x counterparts
000077
000078    *)
000079
000080    TYPE
000081
000082    S4 = STRING[4];
000083
000084    TId = STRING[IDLength];
000085
000086    TButtonMetrics =
000087        RECORD
000088            height:         INTEGER;
000089            curvH:          INTEGER;
000090            curvV:          INTEGER;
000091
```

```
000092          typeStyle:      TTypeStyle;
000093
000094          expandNum:      INTEGER;   {a button's min width is its text's with times this numerator}
000095          expandDen:      INTEGER;   {   ... divided by this denominator}
000096
000097          absMinWidth:    INTEGER;
000098          penState:       PenState;  {for drawing the round-rect}
000099      END;
000100
000101  TStringKey =    RECORD         {Keys for Dialogs in Resource Files}
000102                      trueKey:    LONGINT;
000103                      key:        S4;
000104                  END;
000105
000106
000107
000108  {-------------------------------------------------------------------------------------------------}
000109
000110                                      { ********* CLASSES ********* }
000111
000112  { ------------------------------ classes implemented in file UDialog2 ------------------------------ }
000113
000114
000115  TDialogWindow = SUBCLASS of TDialogBox  {which itself is in UABC}
000116
000117      controlPanel:   TPanel;       {One with a dialogView in it; may be told to push its dflt button}
000118      dialogView:     TDialogView;  {the view installed in SELF.controlPanel}
000119      mainDialog:     TDialog;      {the first dialog installed in SELF.dialogView}
000120
000121     {Creation/Destruction}
000122       FUNCTION TDialogWindow.CREATE(object: TObject; heap: THeap; itsResizability: BOOLEAN;
000123          itsHeight: INTEGER; itsKeyResponse, itsMenuResponse, itsDownInMainWindowResponse: TDiResponse)
000124                                    : TDialogWindow;
000125
000126     {Showing and Hiding}
000127      PROCEDURE TDialogWindow.Appear;  OVERRIDE;
000128      PROCEDURE TDialogWindow.BeDismissed; OVERRIDE;
000129      FUNCTION  TDialogWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;  OVERRIDE;
000130      PROCEDURE TDialogWindow.Disappear; OVERRIDE;
000131
000132     {Commands}
000133      FUNCTION  TDialogWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
000134
000135    END;  {TDialogWindow interface}
000136
000137
000138  {-------------------------------------------------------------------------------------------------}
000139
```

```
000140
000141  TDialogView = SUBCLASS OF TView  {a view which contains dialog images as well as, possibly, other things}
000142
000143      rootDialog:         TDialog;    {The children of this object are the constituent Dialogs of this view}
000144
000145      nonDialogExtent:    LRect;      {intinsic overall extent, dialog + non-dialog actually}
000146
000147      currentDialogImage: TDialogImage; {which descendent owns the mouse during drag}
000148
000149      defaultButton:      TButton;    {which if any button is the default}
000150      hitButton:          TButton;    {which Button was last chosen}
000151      isShowing:          BOOLEAN;    {used to suppress meaningless screen actions for not-yet-showing box}
000152
000153      paintFreeBoxes:     BOOLEAN;    {whether free-checkboxes are to be painted in one sense only}
000154      paintSense:         BOOLEAN;    { ... and if so, in which sense }
000155      startedPainting:    BOOLEAN;    {whether we've begun to paint and hence established paintSense}
000156
000157      styleSheet:         TStyleSheet; {for use by text images seen in the view}
000158
000159      mouseIsDown:        BOOLEAN;
000160      magnetCursor:       TCursorNumber; {to force CursorAt to return this value until mouseIsDown is FALSE}
000161
000162  { *** Public Interface *** }
000163    {Creation/Destruction}
000164      FUNCTION  TDialogView.CREATE(object: TObject; heap: THeap;  itsExtentLRect: LRect; itsPanel: TPanel;
000165             itsPrintManager: TPrintManager; itsRes: Point): TDialogView;
000166      PROCEDURE TDialogView.Free; OVERRIDE;
000167
000168    {Installing, Removing, Activating, Deactivating dialogs}
000169      PROCEDURE TDialogView.AddDialog(dialog: TDialog);
000170      FUNCTION  TDialogView.AddNewDialog(itsKey: S4): TDialog;
000171      PROCEDURE TDialogView.ActivateDialog(dialog: TDialog; whichWay: BOOLEAN);
000172      PROCEDURE TDialogView.RemoveDialog(dialog: TDialog; andFree: BOOLEAN);
000173      PROCEDURE TDialogView.ReplaceDialog(oldDialog, newDialog: TDialog);
000174
000175    {Methods which client should redefine to get a dialogView also to have non-dialog behaviour}
000176      FUNCTION  TDialogView.XCursorAt(mouseLPt: LPoint): TCursorNumber; DEFAULT;
000177      PROCEDURE TDialogView.XDraw; DEFAULT;
000178      PROCEDURE TDialogView.XMousePress(mouseLPt: LPoint); DEFAULT;
000179      PROCEDURE TDialogView.XMouseMove(mouseLPt: LPoint); DEFAULT;
000180      PROCEDURE TDialogView.XMouseRelease; DEFAULT;
000181
000182   {Buttons and checkboxes}
000183      PROCEDURE TDialogView.AbandonThatButton;
000184      PROCEDURE TDialogView.ButtonPushed(button: TButton);   {normally, TDialog's ButtonPushed is used}
000185      PROCEDURE TDialogView.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN);
000186      PROCEDURE TDialogView.PushButton(button: TButton);     {client or ToolKit may call}
000187      PROCEDURE TDialogView.SetDefaultButton(button: TButton);
```

```
000188      {NB:  PushButton sets the dialogView's hitButton to the requested button, assures that it
000189             is highlighted, and then calls the client's ButtonPushed method of the TDialog which
000190             is the parent of the button}
000191
000192  { *** Private Interface ***  (Methods not expected to be redefined or called by client)}
000193      FUNCTION  TDialogView.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000194      PROCEDURE TDialogView.Draw; OVERRIDE;
000195      PROCEDURE TDialogView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
000196      PROCEDURE TDialogView.MouseMove(mouseLPt: LPoint); OVERRIDE;
000197      PROCEDURE TDialogView.MousePress(mouseLPt: LPoint); OVERRIDE;
000198      PROCEDURE TDialogView.MouseRelease; OVERRIDE;
000199      PROCEDURE TDialogView.RecalcExtent; OVERRIDE;
000200
000201  END;  {TDialogView interface}
000202
000203  {-------------------------------------------------------------------------------------------------}
000204
000205
000206  TDialogImage = SUBCLASS OF TImage
000207
000208      parent:         TDialogImage;
000209      isActive:       BOOLEAN;
000210      isEditable:     BOOLEAN;
000211      withID:         BOOLEAN;
000212
000213  {Creation/destruction}
000214      FUNCTION  TDialogImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000215                  itsView: TView; withChildren: BOOLEAN): TDialogImage;
000216
000217      PROCEDURE TDialogImage.ControlHit(control: TDialogImage; toggleDirection: BOOLEAN); DEFAULT;
000218      FUNCTION  TDialogImage.DownAt(mouseLPt: LPoint):  TDialogImage; DEFAULT;
000219      PROCEDURE TDialogImage.Draw; OVERRIDE;
000220      PROCEDURE TDialogImage.DrawJustMe; {called by Draw after children, if any, are told to draw} DEFAULT;
000221      FUNCTION  TDialogImage.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000222      PROCEDURE TDialogImage.PrepareToAppear;
000223      PROCEDURE TDialogImage.RecalcExtent;  OVERRIDE;
000224      FUNCTION  TDialogImage.StillMyMouse(mouseLPt: LPoint): BOOLEAN; DEFAULT;
000225
000226     {The following methods are stubs, redefined in TImageWithID}
000227      PROCEDURE TDialogImage.AddImage(dialogImage: TDialogImage); DEFAULT;
000228      PROCEDURE TDialogImage.ActivateImage(dialogImage: TDialogImage; whichWay: BOOLEAN); DEFAULT;
000229      PROCEDURE TDialogImage.BringToFront(dialogImage: TDialogImage); DEFAULT;
000230      PROCEDURE TDialogImage.ComeForward; DEFAULT;
000231      PROCEDURE TDialogImage.DeleteImage(dialogImage: TDialogImage; andFree: BOOLEAN); DEFAULT;
000232      PROCEDURE TDialogImage.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
000233      FUNCTION  TDialogImage.HasId(id: S255): BOOLEAN; DEFAULT;
000234      FUNCTION  TDialogImage.ObjectWithIDNumber(idNumber: INTEGER):  TDialogImage; DEFAULT;
000235      FUNCTION  TDialogImage.ObjWithId(id: S255): TDialogImage; DEFAULT;
```

```
000236      PROCEDURE TDialogImage.ReplaceImage(replacee, newValue: TDialogImage); DEFAULT;
000237
000238      END;
000239
000240
000241  TImageWithID = SUBCLASS OF TDialogImage   {same interface as TDialogImage, basically}
000242
000243      children:   TList;  {of TDialogImage}
000244      id:         TId;
000245      idNumber:   INTEGER;
000246
000247      FUNCTION  TImageWithID.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000248                  itsView: TView; withChildren: BOOLEAN): TImageWithID;
000249      FUNCTION  TImageWithID.Clone(heap: THeap): TObject;   OVERRIDE;
000250      PROCEDURE TImageWithID.Free; OVERRIDE;
000251
000252      PROCEDURE TImageWithID.AddImage(dialogImage: TDialogImage); OVERRIDE;
000253      PROCEDURE TImageWithID.ActivateImage(dialogImage: TDialogImage; whichWay: BOOLEAN); OVERRIDE;
000254      PROCEDURE TImageWithID.BringToFront(dialogImage: TDialogImage); OVERRIDE;
000255      FUNCTION  TImageWithID.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000256      PROCEDURE TImageWithID.DeleteImage(dialogImage: TDialogImage; andFree: BOOLEAN); OVERRIDE;
000257      PROCEDURE TImageWithID.Draw; OVERRIDE;
000258      PROCEDURE TImageWithID.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
000259      PROCEDURE TImageWithID.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
000260      FUNCTION  TImageWithID.HasId(id: S255): BOOLEAN; OVERRIDE;
000261      PROCEDURE TImageWithID.HaveView(view: TView); OVERRIDE;
000262      FUNCTION  TImageWithID.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000263      FUNCTION  TImageWithID.ObjectWithIDNumber(idNumber: INTEGER):  TDialogImage; OVERRIDE;
000264      FUNCTION  TImageWithID.ObjWithId(id: S255):  TDialogImage; OVERRIDE;
000265      PROCEDURE TImageWithID.OffSetBy(deltaLPt: LPoint); OVERRIDE;
000266      PROCEDURE TImageWithID.RecalcExtent;   OVERRIDE;
000267      PROCEDURE TImageWithID.ReplaceImage(replacee, newValue: TDialogImage); OVERRIDE;
000268      FUNCTION  TImageWithID.StillMyMouse(mouseLPt : LPoint): BOOLEAN; OVERRIDE;
000269
000270      END;
000271
000272  {-------------------------------------------------------------------------------------------}
000273
000274
000275  TDialog = SUBCLASS OF TImageWithID
000276
000277      stringKey:  TStringKey;   {essentially a unique 4-character ID by which this dialog is known}
000278
000279  {Creation}
000280
000281      FUNCTION  TDialog.CREATE(object: TObject; heap: THeap; itsKey: S4; itsView: TView): TDialog;
000282
000283  {Creation of the basic dialog elements:}
```

```
000284
000285    {Elements originating from phrase file; in each case, the text for the legend associated with the
000286          component, if any, as well as a LOCATION for the component, are obtained from the same entry
000287          in the phrase file, with the syntax
000288
000289              <text>@<h-coordinate>,<v-coordinate>
000290
000291           EXAMPLE:  Suppose the following 2 lines are in the Phrase File:
000292
000293                  449
000294                  Next@430,50
000295
000296          If we call NewButton(449, ...), then a button is created, with the text 'Next' inside it;
000297          the button is given idNumber 449, and is located at (430, 50)}
000298
000299  {************************** PUBLIC INTERFACE -- USE THESE METHODS ***********************************}
000300
000301      FUNCTION  TDialog.NewButton(itsPhrase: INTEGER; itsMetrics: TButtonMetrics; sameSizedButton: TButton;
000302                      itsCmdNumber: TCmdNumber): TButton;
000303
000304      FUNCTION  TDialog.NewCluster(itsPhrase: INTEGER): TCluster;
000305
000306      FUNCTION  TDialog.NewFreeCheckbox(itsPhrase: INTEGER;  boxWidth: INTEGER;
000307          boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckBox;
000308
000309      FUNCTION  TDialog.NewInputFrame(itsPhrase: INTEGER; promptTypeStyle: TTypeStyle;
000310              inputOffset: Point; inputTypeStyle: TTypeStyle;
000311              maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect: BOOLEAN;
000312              drawHitLRect: BOOLEAN): TInputFrame;
000313
000314      FUNCTION  TDialog.NewLegend(itsPhrase: INTEGER; itsTypeStyle: TTypeStyle): TLegend;
000315
000316      FUNCTION  TDialog.NewRowOfBoxes(itsPhrase: INTEGER; numberOfBoxes: INTEGER;
000317          startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER): TCluster;
000318
000319    {controls}
000320      PROCEDURE TDialog.ButtonPushed(button: TButton); DEFAULT; {client overrides often}
000321      PROCEDURE TDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN); DEFAULT;
000322                                          {client overrides sometimes}
000323      PROCEDURE TDialog.ControlHit(control: TDialogImage; toggleDirection: BOOLEAN); OVERRIDE;
000324      PROCEDURE TDialog.PushButton(button: TButton);     {client or ToolKit may call}
000325      PROCEDURE TDialog.SelectInputFrame(inputFrame: TInputFrame);
000326      PROCEDURE TDialog.SetDefaultButton(button: TButton);
000327
000328
000329  {************************************ PRIVATE INTERFACE *****************************************}
000330
000331  {These methods of TDialog are largely either for internal use of the building block, or maintained for
```

```
000332   backward compatability with earlier versions of the dialog building block}
000333
000334    {"Standard" elements:}
000335      FUNCTION  TDialog.AddStdButton(itsId: S255; itsXLoc, itsYLoc: LONGINT; sameSizedButton: TButton;
000336                                          itsCmdNumber: TCmdNumber): TButton;
000337      PROCEDURE TDialog.AddOKButton(cmdNumber: TCmdNumber);      {OK Button}
000338      PROCEDURE TDialog.AddCancelButton(cmdNumber: TCmdNumber);  {Cancel Button}
000339      FUNCTION  TDialog.AddStdCluster(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCluster;
000340      FUNCTION  TDialog.AddStdFreeCheckbox(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCheckBox;
000341      FUNCTION  TDialog.AddStdInputFrame(itsId: S255; itsXLoc: LONGINT;
000342                                          itsYLoc: LONGINT; maxInputChars : INTEGER): TInputFrame;
000343      FUNCTION  TDialog.AddStdLegend(itsId: S255; itsXLoc, itsYLoc: LONGINT;
000344                                          itsTypeStyle: TTypeStyle): TLegend;
000345      FUNCTION  TDialog.AddSysLegend(itsId: S255; itsXLoc, itsYLoc: LONGINT): TLegend;
000346
000347    {General creation of dialogImages}
000348      FUNCTION  TDialog.AddButton(itsId: S255; itsLocation: LPoint; itsMetrics: TButtonMetrics;
000349                                          sameSizedButton: TButton; itsCmdNumber: TCmdNumber): TButton;
000350
000351      FUNCTION  TDialog.AddFreeCheckbox(itsID: S255; itsXLoc, itsYLoc: LONGINT; boxWidth: INTEGER;
000352          boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox;
000353
000354      FUNCTION  TDialog.AddBigFreeCheckbox(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCheckbox;
000355
000356      FUNCTION  TDialog.AddRowOfBoxes(itsID: S255; itsXLoc, itsYLoc: LONGINT; numberOfBoxes: INTEGER;
000357          startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER): TCluster;
000358
000359      FUNCTION  TDialog.AddInputFrame(itsId: S255;
000360              promptLocation: LPoint; promptTypeStyle: TTypeStyle;
000361              inputLocation:  LPoint; inputTypeStyle: TTypeStyle;
000362              maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect: BOOLEAN;
000363              drawHitLRect: BOOLEAN): TInputFrame;
000364
000365      FUNCTION  TDialog.DownAt(mouseLPt: LPoint): TDialogImage; OVERRIDE;
000366      PROCEDURE TDialog.RecalcExtent; OVERRIDE;
000367
000368      END;
000369
000370
000371   {-------------------------------------------------------------------------------------------------}
000372
000373
000374   TButton = SUBCLASS OF TImageWithID
000375
000376      cmdNumber:              TCmdNumber;
000377      minWidth:               INTEGER;
000378      isHighlighted:          BOOLEAN;
000379      nextSameSizedButton:    TButton;
```

```
000380      legend:                 TLegend;
000381      buttonMetrics:          TButtonMetrics;
000382
000383  {Creation/Destruction}
000384      FUNCTION  TButton.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
000385           itsLocation: LPoint; itsMetrics: TButtonMetrics;  sameSizedButton: TButton;
000386           itsCmdNumber: TCmdNumber):  TButton;
000387
000388      PROCEDURE TButton.DrawJustMe; OVERRIDE;
000389      PROCEDURE TButton.Highlight(highTransit: THighTransit);
000390      FUNCTION  TButton.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000391      PROCEDURE TButton.MousePress(mouseLPt: LPoint); OVERRIDE;
000392      PROCEDURE TButton.MouseRelease; OVERRIDE;
000393      PROCEDURE TButton.RecalcExtent; OVERRIDE;
000394      PROCEDURE TButton.Recompute(minWidth: INTEGER);
000395      FUNCTION  TButton.StillMyMouse(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
000396
000397  END; {TButton interface}
000398
000399  {-------------------------------------------------------------------------------------------------}
000400
000401  TCheckbox = SUBCLASS of TImageWithID
000402
000403      isSelected:   BOOLEAN;
000404
000405      rectImage:    TRectImage;     {also a child}
000406      legend:       TLegend;        {if nonnil, also a child}
000407
000408      FUNCTION TCheckbox.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
000409          itsLocation: LPoint; boxWidth: INTEGER; boxHeight: INTEGER; wantLabel: BOOLEAN;
000410          labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox;
000411
000412      PROCEDURE TCheckbox.ChangeLabel(newS255: S255);
000413      FUNCTION  TCheckbox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000414      PROCEDURE TCheckbox.Draw; OVERRIDE;
000415      FUNCTION  TCheckbox.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000416      PROCEDURE TCheckbox.MousePress(mouseLPt: LPoint); OVERRIDE;
000417      PROCEDURE TCheckbox.Toggle;
000418
000419    END;  {TCheckbox interface}
000420
000421  {-------------------------------------------------------------------------------------------------}
000422
000423  TCluster = SUBCLASS of TImageWithID
000424
000425     {children:   TList;  (of TCheckbox) }
000426
000427      location:   LPoint;               {only used for adding the first aligned checkbox}
```

```
000428      hitBox:     TCheckbox;              {which one was just successfully queried by Hit}
000429      hiLitBox:   TCheckbox;              {which one is highlighted}
000430      lastBox:    TCheckBox;              {the checkbox most recently added checkbox}
000431
000432      FUNCTION  TCluster.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
000433                                itsLocation : LPoint): TCluster;
000434
000435  {****** PUBLIC INTERFACE:
000436   ******
000437   ****** Create a cluster using TDialog.NewCluster; add checkboxes to it by calling any of the following
000438   ****** three methods.  To change which box is selected in the cluster programmatically, call SelectBox
000439   ******
000440   ****** To find out which box is selected in a cluster, look at cluster.hiLitBox.idNumber}
000441
000442       FUNCTION  TCluster.NewAlignedCheckbox(itsPhrase: INTEGER; selectThisOne: BOOLEAN): TCheckbox;
000443       FUNCTION  TCluster.NewCheckbox(itsPhrase: INTEGER; boxWidth: INTEGER;
000444                         boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle;
000445                         selectThisOne: BOOLEAN): TCheckbox;
000446       PROCEDURE TCluster.AddRowOfBoxes(numberOfBoxes: INTEGER; startingIDNumber: INTEGER;
000447                                boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER);
000448
000449      PROCEDURE TCluster.SelectBox(checkbox: TCheckbox); {select this box, deselecting others}
000450
000451  {****** PRIVATE INTERFACE:
000452   ******
000453   ******  These remaining methods of TCluster are for primarily for internal use:}
000454
000455      FUNCTION  TCluster.AddAlignedCheckbox(itsId: S255; selectThisOne: BOOLEAN): TCheckbox;
000456      FUNCTION  TCluster.AddCheckbox(itsID: S255; itsLocation: LPoint; boxWidth: INTEGER;
000457                         boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle;
000458                         selectThisOne: BOOLEAN): TCheckbox;
000459      FUNCTION  TCluster.Hit(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
000460      PROCEDURE TCluster.MousePress(mouseLPt: LPoint); OVERRIDE;
000461      FUNCTION  TCluster.StillMyMouse(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
000462
000463  END; {TCluster interface}
000464
000465  {-------------------------------------------------------------------------------------------------}
000466
000467
000468  TInputFrame = SUBCLASS OF TImageWithID
000469
000470      textDialogImage:   TTextDialogImage;
000471      prompt:            TLegend;
000472
000473      borders:           Rect;
000474
000475      drawInputLRect:    BOOLEAN;  {whether or not to draw a faint box around the input LRect}
```

```
000476      drawHitLRect:      BOOLEAN;  {whether or not to frame the hit rectangle}
000477      maxInputChars:     INTEGER;
000478      inputTypeStyle:    TTypeStyle;
000479
000480      FUNCTION  TInputFrame.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
000481            promptLocation: LPoint; promptTypeStyle: TTypeStyle;
000482            inputLocation:  LPoint; inputTypeStyle: TTypeStyle;  maxInputChars: INTEGER;
000483            itsBorders: Rect; drawInputLRect: BOOLEAN; drawHitLRect: BOOLEAN
000484            ): TInputFrame;
000485
000486
000487  { ***************************************** PUBLIC INTERFACE ************************************* }
000488
000489     {Create an input frame by calling TDialog.NewInputFrame; use GetContents and SupplantContents
000490      to find out what has been typed, and to change what appears in the typing area}
000491
000492      PROCEDURE TInputFrame.GetContents(VAR theStr: S255);     {inspect current frame contents}
000493      PROCEDURE TInputFrame.SupplantContents(newStr: S255);    {change current frame contents}
000494
000495  { ***************************************** PRIVATE INTERFACE ************************************ }
000496
000497      FUNCTION  TInputFrame.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000498      PROCEDURE TInputFrame.Draw; OVERRIDE;
000499      FUNCTION  TInputFrame.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000500      PROCEDURE TInputFrame.MousePress(mouseLPt: LPoint); OVERRIDE;
000501      PROCEDURE TInputFrame.RecalcExtent; OVERRIDE;
000502      FUNCTION  TInputFrame.StillMyMouse(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
000503
000504   END;   {TInputFrame interface}
000505
000506  {-------------------------------------------------------------------------------------------------}
000507
000508  TLegend = SUBCLASS OF TDialogImage
000509
000510          location:           LPoint;
000511          paragraph:          TParagraph;
000512          wouldBeDraggable:   BOOLEAN;        {whether, during layout, it should itself be draggable}
000513          usesSysFont:        BOOLEAN;        {whether it is in system font -- a special case}
000514
000515      FUNCTION  TLegend.CREATE(object: TObject; heap: THeap; itsChars: S255; itsView: TView;
000516          itsLocation: LPoint; itsTypeStyle: TTypeStyle): TLegend;
000517      PROCEDURE TLegend.Free; OVERRIDE;
000518
000519  { ***************************************** PUBLIC INTERFACE ************************************* }
000520
000521      PROCEDURE TLegend.ChangeToPhrase(newPhrase: INTEGER); {for getting new text from phrase file}
000522      PROCEDURE TLegend.ChangeString(newString: S255);      {for getting new text from a string}
000523      PROCEDURE TLegend.GetString(VAR itsString: S255); {determine current chars residing in the legend}
```

```
000524
000525  { ****************************************** PRIVATE INTERFACE ************************************* }
000526      PROCEDURE TLegend.Draw; OVERRIDE;
000527      PROCEDURE TLegend.GetBoxRight;   {sets extent based on current chars & location}
000528      FUNCTION  TLegend.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000529      PROCEDURE TLegend.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000530      PROCEDURE TLegend.RecalcExtent; OVERRIDE;
000531
000532      END;
000533
000534  { ------------------------------ classes implemented in file UDialog3 ------------------------------}
000535
000536
000537
000538  TPicObject = SUBCLASS OF TImageWithID {An Object which holds a QD Picture File} {CAUTION: totally untested}
000539
000540      picture:        PicHandle;
000541      boxAtCreation:  Rect;   {need to get itsView parameter into all these guys}
000542
000543      FUNCTION  TPicObject.CREATE(object: TObject; heap: THeap; itsId: S255;
000544           itsView: TView; itsLocation: LPoint; itsPicHandle: PicHandle): TPicObject;
000545      PROCEDURE TPicObject.Free; OVERRIDE;
000546
000547      PROCEDURE TPicObject.Draw; OVERRIDE;
000548
000549      END;
000550
000551  {-------------------------------------------------------------------------------------------------------}
000552
000553    TRectImage = SUBCLASS OF TDialogImage   {a rectangle packaged as a object}
000554
000555          penState: PenState;
000556
000557      FUNCTION TRectImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000558          itsView: TView; itsPenState: PenState; withChildren: BOOLEAN): TRectImage;
000559
000560      PROCEDURE TRectImage.Draw; OVERRIDE;
000561      FUNCTION  TRectImage.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000562      END;
000563
000564  {-------------------------------------------------------------------------------------------------------}
000565
000566
000567  TTextDialogImage = SUBCLASS OF TImageWithID
000568
000569       textImage:         TTextImage;
000570       wouldBeDraggable:  BOOLEAN;
000571       refCount:          INTEGER;
```

```
000572
000573      FUNCTION  TTextDialogImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000574                    itsView: TView; itsTypeStyle: TTypeStyle;
000575                    itsInitialChars: S255): TTextDialogImage;
000576      PROCEDURE TTextDialogImage.Free; OVERRIDE;
000577
000578      PROCEDURE TTextDialogImage.ChangeRefCountBy(delta: INTEGER);
000579      FUNCTION  TTextDialogImage.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000580      PROCEDURE TTextDialogImage.Draw; OVERRIDE;
000581      FUNCTION  TTextDialogImage.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000582      PROCEDURE TTextDialogImage.MousePress(mouseLPt: LPoint); OVERRIDE;
000583      PROCEDURE TTextDialogImage.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000584       END;
000585
000586  {-------------------------------------------------------------------------------------------------}
000587
000588  TFrameSelection = SUBCLASS OF TSelection  {the phony selection covering TextSelection in an input frame}
000589
000590      inputFrame: TInputFrame;  {the input frame in which the selection occurs}
000591
000592      FUNCTION  TFrameSelection.CREATE(object: TObject; heap: THeap; itsInputFrame: TInputFrame)
000593                                    : TFrameSelection;
000594
000595      FUNCTION  TFrameSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; OVERRIDE;
000596      PROCEDURE TFrameSelection.KeyChar(ch: CHAR); OVERRIDE;
000597      PROCEDURE TFrameSelection.KeyEnter(dh, dv: INTEGER); OVERRIDE;
000598      PROCEDURE TFrameSelection.KeyReturn; OVERRIDE;
000599      PROCEDURE TFrameSelection.KeyTab(fBackward: BOOLEAN);  OVERRIDE;
000600      PROCEDURE TFrameSelection.MousePress(mouseLPt: LPoint);  OVERRIDE;
000601      PROCEDURE TFrameSelection.PerformCommand(command: TCommand; cmdPhase: TCmdPhase); OVERRIDE;
000602      PROCEDURE TFrameSelection.Restore; OVERRIDE;
000603
000604  END;  {TFrameSelection interface}
000605
000606
000607  TPlannerView = SUBCLASS OF TDialogView  {a view within which images are laid out}
000608
000609    {Variables}
000610
000611      viewBeingPlanned:   TView;
000612
000613      allowSketching:     BOOLEAN;   {for internal use of the layout mechanism}
000614      retainPickedBox:    BOOLEAN;
000615      currentLayoutBox:   TLayoutBox;
000616
000617    {Creation/Destruction}
000618      FUNCTION  TPlannerView.CREATE(object: TObject; heap: THeap; itsViewBeingPlanned: TView;
000619                 itsPanel: TPanel; itsAllowSketching: BOOLEAN; itsRetainPickedBox: BOOLEAN): TPlannerView;
```

```
000620      PROCEDURE TPlannerView.Init(itsListOfImages: TList);
000621      FUNCTION  TPlannerView.NewLayoutBox(image: TImage): TLayoutBox; {return NIL if element not to be shown}
000622
000623      PROCEDURE TPlannerView.Free; OVERRIDE;
000624
000625    {Display}
000626      PROCEDURE TPlannerView.Draw; OVERRIDE;
000627
000628    {Mouse Tracking}
000629      FUNCTION  TPlannerView.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000630      PROCEDURE TPlannerView.MouseMove(mouseLPt: LPoint); OVERRIDE;
000631      PROCEDURE TPlannerView.MousePress(mouseLPt: LPoint); OVERRIDE;
000632      PROCEDURE TPlannerView.MouseRelease; OVERRIDE;
000633
000634    {Enumeration of components}
000635      PROCEDURE TPlannerView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
000636
000637      END;
000638
000639  {-------------------------------------------------------------------------------------------------}
000640
000641
000642      TLayoutBox = SUBCLASS OF TImageWithID
000643
000644          {Variables}
000645          manipulee:                  TImage;
000646          titleTab:                   TTitleTab;
000647
000648          suppressDrawingManipulee:   BOOLEAN;
000649
000650          isResizable:                BOOLEAN;
000651          borders:                    Rect;
000652          wouldMakeSelection:         BOOLEAN;   {client must directly set if not wanting default 'FALSE'}
000653
000654          isDraggable:                BOOLEAN;
000655          shouldFrame:                BOOLEAN;
000656
000657          hasDraggee:                 BOOLEAN;
000658
000659        {Creation/Destruction}
000660          FUNCTION  TLayoutBox.CREATE(object: TObject; heap: THeap; baseExtent: LRect; itsID: S255;
000661              itsParent: TLayoutBox; itsView: TView; itsManipulee: TImage; itsBorders: Rect;
000662              itsResizable: BOOLEAN; itsSuppression: BOOLEAN; withChildren: BOOLEAN): TLayoutBox;
000663          PROCEDURE TLayoutBox.Free; OVERRIDE;
000664
000665        {Change and Display}
000666          PROCEDURE TLayoutBox.ChangeDragState(enteringDrag: BOOLEAN);
000667          PROCEDURE TLayoutBox.ConsiderMouse(mouseLPt: LPoint; VAR madeSelection: BOOLEAN;
```

```
000668                                          VAR pickedLayoutBox: TLayoutBox); DEFAULT;
000669          FUNCTION  TLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000670          PROCEDURE TLayoutBox.Draw; OVERRIDE;
000671          PROCEDURE TLayoutBox.DrawJustMe; OVERRIDE;
000672          PROCEDURE TLayoutBox.FreeManipulee;
000673          PROCEDURE TLayoutBox.Highlight(highTransit: THighTransit);
000674          PROCEDURE TLayoutBox.MousePress(mouseLPT: LPoint); OVERRIDE;
000675          PROCEDURE TLayoutBox.Move(deltaLPt: LPoint); DEFAULT;
000676          FUNCTION  TLayoutBox.NoTitleTab(heap: THeap): TTitleTab;
000677          PROCEDURE TLayoutBox.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000678          PROCEDURE TLayoutBox.OffsetLayoutBoxBy(deltaLPt: LPoint; textImageAsWell: BOOLEAN); DEFAULT;
000679          PROCEDURE TLayoutBox.RecalcExtent; OVERRIDE;
000680          PROCEDURE TLayoutBox.Resize(newExtent: LRect); OVERRIDE;
000681          PROCEDURE TLayoutBox.TabGrabbed; DEFAULT;
000682
000683          END;
000684
000685      TLegendLayoutBox = SUBCLASS OF TLayoutBox  {manipulee is a TLegend}
000686
000687          textDialogImage:  TTextDialogImage;
000688
000689        {Creation/Destruction}
000690          FUNCTION  TLegendLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView; itsLegend: TLegend
000691                  ): TLegendLayoutBox;
000692
000693          FUNCTION  TLegendLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000694          PROCEDURE TLegendLayoutBox.Draw; OVERRIDE;
000695          PROCEDURE TLegendLayoutBox.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000696          PROCEDURE TLegendLayoutBox.OffsetLayoutBoxBy(deltaLPt: LPoint; textImageAsWell: BOOLEAN); OVERRIDE;
000697            {use of the second argument is strange and non self-explanatory; comments in the internal
000698             documentation may help.  Nobody should be calling this old boy from outside, anyway}
000699          PROCEDURE TLegendLayoutBox.MousePress(mouseLPT: LPoint); OVERRIDE;
000700          PROCEDURE TLegendLayoutBox.RecalcExtent; OVERRIDE;
000701
000702          END;
000703
000704
000705      TButtonLayoutBox = SUBCLASS OF TLayoutBox  {manipulee is a TButton}
000706
000707        {Variables}
000708          nextSameSizedBox:   TButtonLayoutBox;
000709          oldLegendTopLeft:   LPoint;
000710
000711        {Creation/Destruction}
000712          FUNCTION  TButtonLayoutBox.CREATE(object: TObject; heap: THeap; itsButton: TButton;
000713                                          itsView: TView): TButtonLayoutBox;
000714
000715        {Other Methods}
```

```
000716          PROCEDURE TButtonLayoutBox.ConsiderMouse(mouseLPt: LPoint; VAR madeSelection: BOOLEAN;
000717                                              VAR pickedLayoutBox: TLayoutBox); OVERRIDE;
000718          FUNCTION  TButtonLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000719          PROCEDURE TButtonLayoutBox.DrawJustMe; OVERRIDE;
000720          PROCEDURE TButtonLayoutBox.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000721          PROCEDURE TButtonLayoutBox.RecalcExtent; OVERRIDE;
000722          PROCEDURE TButtonLayoutBox.RecalcJustMe;
000723
000724          END;
000725
000726
000727      TTitleTab = SUBCLASS OF TImage
000728
000729          layoutBox:          TLayoutBox;
000730          legend:             TLegend;
000731          shouldDrawLegend:   BOOLEAN;  {FALSE if string is too wide to fit}
000732
000733          FUNCTION  TTitleTab.CREATE(object: TObject; heap: THeap; itsLayoutBox: TLayoutBox; itsHeight: INTEGER;
000734              itsCaption: S255): TTitleTab;
000735          PROCEDURE TTitleTab.Free; OVERRIDE;
000736
000737          PROCEDURE TTitleTab.Draw; OVERRIDE;
000738          PROCEDURE TTitleTab.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000739          PROCEDURE TTitleTab.Resize(newExtent: LRect); OVERRIDE;
000740          END;
000741
000742
000743      TLayPickSelection = SUBCLASS OF TSelection
000744
000745        {Variables}
000746          layoutBox:  TLayoutBox;
000747
000748          FUNCTION  TLayPickSelection.CREATE(object: TObject; heap: THeap; itsView: TPlannerView;
000749                          itsKind: INTEGER; itsLayoutBox: TLayoutBox; itsAnchorLPt: LPoint): TLayPickSelection;
000750
000751          FUNCTION  TLayPickSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
000752                                                  : BOOLEAN; OVERRIDE;
000753          PROCEDURE TLayPickSelection.Deselect; OVERRIDE;
000754          PROCEDURE TLayPickSelection.Highlight(highTransit: THighTransit); OVERRIDE;
000755          PROCEDURE TLayPickSelection.KeyClear; OVERRIDE;
000756          PROCEDURE TLayPickSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
000757          PROCEDURE TLayPickSelection.MouseRelease; OVERRIDE;
000758          PROCEDURE TLayPickSelection.Restore; OVERRIDE;
000759
000760          END;
000761
000762
000763      TLayMoveCmd = SUBCLASS OF TCommand
```

```
000764
000765        {Variables}
000766          layoutBox:  TLayoutBox;
000767
000768          hOffset:    LONGINT;
000769          vOffset:    LONGINT;
000770
000771        {Creation}
000772          FUNCTION  TLayMoveCmd.CREATE(object: TObject; heap: THeap; itsLayoutBox: TLayoutBox;
000773                                       itsHOffset, itsVOffset: LONGINT): TLayMoveCmd;
000774
000775        {Command Execution}
000776          PROCEDURE  TLayMoveCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
000777
000778          END;
000779
000780      TEditLegendSelection = SUBCLASS OF TSelection
000781
000782        {Variables}
000783         legendLayoutBox: TLegendLayoutBox;
000784         hostLegend:      TLegend;
000785         textDialogImage: TTextDialogImage;
000786         suppressHost:    BOOLEAN;
000787         tripleClick:     BOOLEAN; {+SW+}
000788
000789        {Creation/Destruction}
000790         FUNCTION  TEditLegendSelection.CREATE(object: TObject; heap: THeap; itsLegendLayoutBox:
000791                                               TLegendLayoutBox; itsAnchorLPt: LPoint): TEditLegendSelection;
000792         FUNCTION  TEditLegendSelection.Clone(heap: THeap): TObject;  OVERRIDE;
000793         PROCEDURE TEditLegendSelection.Deselect; OVERRIDE;
000794         PROCEDURE TEditLegendSelection.Free; OVERRIDE;
000795
000796        {Udders}
000797         FUNCTION  TEditLegendSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
000798                                                    : BOOLEAN; OVERRIDE;
000799         PROCEDURE TEditLegendSelection.KeyBack(fWord: BOOLEAN); OVERRIDE;
000800         PROCEDURE TEditLegendSelection.KeyChar(ch: CHAR); OVERRIDE;
000801         PROCEDURE TEditLegendSelection.KeyEnter(dh, dv: INTEGER); OVERRIDE;
000802         PROCEDURE TEditLegendSelection.KeyReturn; OVERRIDE;
000803         PROCEDURE TEditLegendSelection.MouseMove(mouseLPt: LPoint); OVERRIDE; {+SW+}
000804         PROCEDURE TEditLegendSelection.MousePress(mouseLPt: LPoint); OVERRIDE; {+SW+}
000805         PROCEDURE TEditLegendSelection.MouseRelease; OVERRIDE; {+SW+}
000806         FUNCTION  TEditLegendSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
000807         PROCEDURE TEditLegendSelection.PerformCommand(command: TCommand; cmdPhase: TCmdPhase);  OVERRIDE;
000808         PROCEDURE TEditLegendSelection.Restore; OVERRIDE;
000809         PROCEDURE TEditLegendSelection.Reveal(asMuchAsPossible: BOOLEAN);  OVERRIDE;
000810
000811         END;
```

```
000812
000813
000814      TDialogDesignWindow = SUBCLASS OF TDialogWindow
000815
000816          hostWindow:         TWindow;
000817          hostDialogView:     TDialogView;
000818          fromDialogBox:      BOOLEAN;
000819
000820       FUNCTION  TDialogDesignWindow.CREATE(object: TObject; heap: THeap;
000821                      itsHostDialogView: TDialogView): TDialogDesignWindow;
000822
000823       FUNCTION  TDialogDesignWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
000824                                              : BOOLEAN; OVERRIDE;
000825       FUNCTION  TDialogDesignWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
000826       PROCEDURE TDialogDesignWindow.RelinquishControl;
000827       PROCEDURE TDialogDesignWindow.Resize(moving: BOOLEAN); OVERRIDE;
000828       PROCEDURE TDialogDesignWindow.SeizeControl;
000829
000830       END;
000831
000832
000833
000834  { ------------------------------ classes implemented in file UDialog4 -----------------------------}
000835
000836
000837  TStdPrintManager = SUBCLASS OF TPrintManager
000838
000839      FUNCTION  TStdPrintManager.CREATE(object: TObject; heap: THeap): TStdPrintManager;
000840
000841      PROCEDURE TStdPrintManager.EnterPageEditting; OVERRIDE;
000842      PROCEDURE TStdPrintManager.Init(itsMainView: TView; itsDfltMargins: LRect); OVERRIDE;
000843      PROCEDURE TStdPrintManager.ReactToPrinterChange; OVERRIDE;
000844      PROCEDURE TStdPrintManager.SetDfltHeadings;  OVERRIDE;
000845
000846      END;
000847
000848  TLegendHeading = SUBCLASS OF THeading
000849
000850      masterLegend:   TLegend;
000851      currentLegend:  TLegend;
000852
000853      topToBaseline:  INTEGER;  {offset from box top to baseline}
000854      borders:        Rect;    {size by which box exceeds legend's extent}
000855
000856    {Creation/Destruction}
000857      FUNCTION  TLegendHeading.CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
000858          itsString: S255; itsTypeStyle: TTypeStyle;
000859          itsPageAlignment: TPageAlignment; itsOffsetFromAlignment: LPoint;
```

```
000860          itsBorders: Rect): TLegendHeading;
000861      PROCEDURE  TLegendHeading.Free; OVERRIDE;
000862
000863    {Nyingine}
000864      PROCEDURE TLegendHeading.AdjustForPage(pageNumber: LONGINT; editing: BOOLEAN); OVERRIDE;
000865      PROCEDURE TLegendHeading.Draw; OVERRIDE;
000866      FUNCTION  TLegendHeading.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
000867      PROCEDURE TLegendHeading.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000868      PROCEDURE TLegendHeading.RecalcExtent; OVERRIDE;
000869      FUNCTION  TLegendHeading.ShouldFrame: BOOLEAN; OVERRIDE;
000870
000871      END;
000872
000873
000874  TPageDesignWindow = SUBCLASS OF TDialogWindow
000875
000876      hostView:       TView;  {the view whose pages are being designed in this dialog}
000877      layoutPanel:    TPanel; {my controlPanel is the status panel}
000878
000879      FUNCTION  TPageDesignWindow.CREATE(object: TObject; heap: THeap; itsHostView: TView): TPageDesignWindow;
000880
000881      PROCEDURE TPageDesignWindow.Disappear;  OVERRIDE;
000882      FUNCTION  TPageDesignWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;  OVERRIDE;
000883
000884      END;
000885
000886
000887  TPagePlannerView = SUBCLASS OF TPlannerView
000888
000889      FUNCTION TPagePlannerView.CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
000890                      itsPanel: TPanel): TPagePlannerView;
000891
000892      PROCEDURE TPagePlannerView.Draw; OVERRIDE;
000893
000894      END;
000895
000896
000897  TPageStatusDialog = SUBCLASS OF TDialog
000898
000899      currentHeading:     THeading;
000900
000901      oddEvenCluster:     TCluster;
000902      minPageFrame:       TInputFrame;
000903      maxPageFrame:       TInputFrame;
000904      alignCluster:       TCluster;
000905      unitsCluster:       TCluster;
000906      marginTitle:        TLegend;
000907
```

```
000908      leftCluster:        TCluster;
000909      topCluster:         TCluster;
000910      rightCluster:       TCluster;
000911      bottomCluster:      TCluster;
000912
000913    {Creation/Destruction}
000914      FUNCTION  TPageStatusDialog.CREATE(object: TObject; heap: THeap; itsPanel: TPanel): TPageStatusDialog;
000915
000916    {Sonst}
000917      PROCEDURE TPageStatusDialog.ButtonPushed(button: TButton); OVERRIDE;
000918      PROCEDURE TPageStatusDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN); OVERRIDE;
000919      FUNCTION  TPageStatusDialog.DownAt(mouseLPt: LPoint): TDialogImage; OVERRIDE;
000920      PROCEDURE TPageStatusDialog.Draw; OVERRIDE;
000921      PROCEDURE TPageStatusDialog.InspectHeadingParms(VAR oddOnly, evenOnly: BOOLEAN;
000922                                      VAR pageAlignment: TPageAlignment; VAR minPage, maxPage: LONGINT);
000923      PROCEDURE TPageStatusDialog.SetHeadingParms(oddOnly, evenOnly: BOOLEAN;
000924                                      pageAlignment: TPageAlignment;  minPage, maxPage: LONGINT);
000925      END;
000926
000927
000928  TPageLayoutBox = SUBCLASS OF TLayoutBox
000929
000930
000931    {Creation/Destruction}
000932      FUNCTION  TPageLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView; itsHeading: THeading;
000933                      itsResizable: BOOLEAN): TPageLayoutBox;
000934
000935      PROCEDURE TPageLayoutBox.FreeManipulee; OVERRIDE;
000936      PROCEDURE TPageLayoutBox.TabGrabbed; OVERRIDE;
000937      END;
000938
000939
000940  TLgHdngLayoutBox = SUBCLASS OF TPageLayoutBox
000941
000942      legendLayoutBox: TLegendLayoutBox;
000943
000944      FUNCTION  TLgHdngLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView;
000945          itsLegendHeading: TLegendHeading): TLgHdngLayoutBox;
000946
000947      FUNCTION  TLgHdngLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
000948      PROCEDURE TLgHdngLayoutBox.Draw; OVERRIDE;
000949      PROCEDURE TLgHdngLayoutBox.MousePress(mouseLPT: LPoint); OVERRIDE;
000950      PROCEDURE TLgHdngLayoutBox.Move(deltaLPt: LPoint); OVERRIDE;
000951      PROCEDURE TLgHdngLayoutBox.RecalcExtent; OVERRIDE;
000952
000953      END;
000954
000955
```

```
000956  VAR
000957      stdFrameBorders:        Rect;           {extra space around an input-frame and its text}
000958      stdHdngBorders:         Rect;           {extra space around a standard heading}
000959      stdHdngTypeStyle:       TTypeStyle;     {tile 12 monospaced, normal faces, for titles}
000960      stdIDBorders:           Rect;           {a title tab with string, and a small border on the other 3 sides}
000961      stdInputTypeStyle:      TTypeStyle;     {std input font/faces}
000962      stdFrmeOffset:          Point;          {std distance between input frame's prompt and input rect}
000963      stdLabelOffset:         Point;          {offset from top-left corner of a checkbox to leftmost pt of
000964                                               baseline of label}
000965      stdPlainBorders:        Rect;           {a slim captionless title tab, and a small border on the other
000966                                               3 sides}
000967      stdThinBorders:         Rect;           {a slim captionless title tab above; no other borders}
000968      titleTypeStyle:         TTypeStyle;     {tile 15 monospaced, for titles of layout boxes}
000969      {NB: All the above are initialized in the creation block of TDialogWindow}
000970
000971      stdButtonMetrics:       TButtonMetrics; {reinitialized in TDialog.CREATE each time}
000972
000973
000974  {Unit-Global Procedures}
000975
000976  FUNCTION  NewStdDialogWindow(heap: THeap; itsHeight: INTEGER; itsKeyResponse, itsMenuResponse,
000977                 itsDownInMainWindowResponse: TDiResponse): TDialogWindow;
000978              {sets up a standard, nonresizable, dialogWindow, and installs a single Panel into it, into
000979               which it installs a single DialogView}
000980
000981  FUNCTION  NewStdLegend(heap: THeap; itsChars: S255; itsXLoc, itsYLoc: LONGINT; itsView: TView;
000982                         itsTypeStyle: TTypeStyle): TLegend;
000983
000984  FUNCTION  NewSysLegend(heap: THeap; itsChars: S255; itsXLoc, itsYLoc: LONGINT; itsView: TView): TLegend;
000985
000986  PROCEDURE SetParaExtent(paragraph: TParagraph; view: TView; location: LPoint; VAR extentLRect: LRect);
000987
000988  PROCEDURE LRectAddBorders(baseLRect: LRect; borders: Rect; VAR resultLRect: LRect);
000989
000990  PROCEDURE GetTextAndLocation(phraseNumber: INTEGER; VAR itsChars: S255; VAR itsLocation: LPoint);
000991
000992
000993  IMPLEMENTATION
000994
000995  {$I LIBTK/UDialog2}    {dialogs}
000996  {$I LIBTK/UDialog3}    {layout}
000997  {$I LIBTK/UDialog4}    {page margins}
000998
000999  (**********
001000  {$I UDialog2}    {dialogs}
001001  {$I UDialog3}    {layout}
001002  {$I UDialog4}    {page margins}
001003  **********)
```

```
001004
001005  END.  {unit UDialog}
001006

End of File -- Lines: 1006 Characters: 43015
```

```
================================================================================================
FILE: "LIBTK/UDIALOG2.TEXT"
================================================================================================

000001  (*          Copyright 1984 by Apple Computer, Inc
000002
000003  UDialog2
000004
000005  TDialogWindow  -- TDialogView  -- TDialogImage  -- TImageWithID  -- TDialog  --
000006  TButton  -- TCheckbox  -- TCluster  -- TInputFrame  -- TLegend
000007
000008  *)
000009
000010  {04/23/84 1210  SetParaExtent uses thePad rather than view's screenPad if amPrinting}
000011  {04/23/84 1210  Removed all references to 'underEdit' field of TDialogImage}
000012  {04/15/84 2345  Spring Release latest}
000013  {04/04/84 2300  Spring Prelim Release}
000014  {01/29/84 1750  RELEASE TK8D}
000015  {12/22/83 1927  RELEASE TK8A}
000016
000017  {$IFC fRngABC}
000018  {$R+}
000019  {$ELSEC}
000020  {$R-}
000021  {$ENDC}
000022
000023  {$IFC fSymABC}
000024  {$D+}
000025  {$ELSEC}
000026  {$D-}
000027  {$ENDC}
000028
000029  VAR copyRight: STRING[45];
000030
000031  {-------------------------------------------------------------------------------------------------}
000032
000033  {$S DlgAlloc}
000034  PROCEDURE GetTextAndLocation(phraseNumber: INTEGER; VAR itsChars: S255; VAR itsLocation: LPoint);
000035      VAR rawPhrase:  S255;
000036          restOfIt:   S255;
000037          morsel:     S255;
000038          semiColon:  INTEGER;
000039          comma:      INTEGER;
000040          aLocation:  LPoint;
000041          FUNCTION OKIntegerValue(Str: S255; VAR itsValue: LONGINT): BOOLEAN;
000042                  VAR result: TConvResult;
000043              BEGIN
```

```
000044              StrToLInt(@str, itsValue, result);
000045              OkIntegerValue := (result = cvValid);
000046              END;
000047      BEGIN  {someone please optimize this someday}
000048      {$IFC fTrace}BP(11);{$ENDC}
000049      process.GetAlert(phraseNumber, rawPhrase);
000050      semiColon := POS('@',rawPhrase);
000051      IF semiColon = 0 THEN
000052          semiColon := LENGTH(rawPhrase) + 1;
000053      itsChars := COPY(rawPhrase, 1, semiColon - 1);
000054      restOfIt := COPY(rawPhrase, semiColon + 1, LENGTH(rawPhrase) - semiColon);
000055      comma := POS( ',', restOfIt);
000056
000057      morsel := COPY(restOfIt, 1, comma - 1);
000058      IF OKIntegerValue(morsel, aLocation.h) THEN
000059          BEGIN
000060          morsel := COPY(restOfIt, comma + 1, LENGTH(restOfIt) - comma);
000061          IF NOT OKIntegerValue(morsel, aLocation.v) THEN
000062              aLocation.v := 100;
000063          END
000064      ELSE
000065          SetLPt(aLocation, 100, 100);
000066
000067      itsLocation := aLocation;
000068      {$IFC fTrace}EP;{$ENDC}
000069      END;
000070
000071
000072  {$S TK2Start}
000073  PROCEDURE LRectAddBorders(baseLRect: LRect; borders: Rect; VAR resultLRect: LRect);
000074      BEGIN
000075      {$IFC fTrace}BP(11);{$ENDC}
000076      resultLRect.left := baseLRect.left + borders.left;
000077      resultLRect.top := baseLRect.top + borders.top;
000078      resultLRect.right := baseLRect.right + borders.right;
000079      resultLRect.bottom := baseLRect.bottom + borders.bottom;
000080      {$IFC fTrace}EP;{$ENDC}
000081      END;
000082
000083
000084  {$S TK2Start}
000085  {"temporary" implementation, slow, unwieldy}
000086  PROCEDURE SetParaExtent(paragraph: TParagraph; view: TView; location: LPoint; VAR extentLRect: LRect);
000087      VAR extent:     Rect;
000088          lExtent:    LRect;
000089          pad:        TPad; {+SW+}
000090      BEGIN
000091          {$IFC fTrace}BP(11);{$ENDC}
```

```
000092          paragraph.BuildExtentLRect(zeroLPt, lExtent); {assumes grafPort device is SCREEN for textWidth}
000093          noPad.LRectToRect(lExtent, extent);
000094  (*      view.screenPad.RectToLRect(extent, extentLRect);  *)
000095          IF amPrinting THEN
000096              pad := thePad
000097          ELSE
000098              pad := view.screenPad;
000099          pad.RectToLRect(extent, extentLRect);   {+SW+}
000100          OffsetLRect(extentLRect, location.h, location.v);
000101          {$IFC fTrace}EP;{$ENDC}
000102      END;
000103
000104
000105  {$S DlgAlloc}
000106  FUNCTION  NewStdDialogWindow(heap: THeap; itsHeight: INTEGER; itsKeyResponse,
000107                  itsMenuResponse, itsDownInMainWindowResponse: TDiResponse):  TDialogWindow;
000108      VAR dialogWindow:   TDialogWindow;
000109          panel:          TPanel;
000110          dialogView:     TDialogView;
000111          extentLRect:    LRect;
000112      BEGIN
000113          {$IFC fTrace}BP(11);{$ENDC}
000114          dialogWindow := TDialogWindow.CREATE(NIL, heap, FALSE {not resizable}, itsHeight,
000115                  itsKeyResponse, itsMenuResponse, itsDownInMainWindowResponse);
000116          panel := TPanel.CREATE(NIL, heap, dialogWindow, 0, screenBits.bounds.right, [], []);
000117          dialogWindow.controlPanel := panel;
000118
000119          SetLRect(extentLRect, 0, 0, screenBits.bounds.right, screenBits.bounds.bottom - 40);
000120          dialogView := TDialogView.CREATE(NIL, heap, extentLRect, panel, NIL, screenRes);
000121          dialogWindow.dialogView := dialogView;
000122          NewStdDialogWindow := dialogWindow;
000123          {$IFC fTrace}EP;{$ENDC}
000124      END;
000125
000126
000127  {$S DlgAlloc}
000128      FUNCTION  NewSysLegend(heap: THeap; itsChars: S255; itsXLoc, itsYLoc: LONGINT; itsView: TView): TLegend;
000129      BEGIN
000130          {$IFC fTrace}BP(11);{$ENDC}
000131          NewSysLegend := NewStdLegend(heap, itsChars, itsXLoc, itsYLoc, itsView, sysTypeStyle);
000132          {$IFC fTrace}EP;{$ENDC}
000133      END;
000134
000135
000136  {$S DlgAlloc}
000137      FUNCTION  NewStdLegend(heap: THeap; itsChars: S255; itsXLoc, itsYLoc: LONGINT; itsView: TView;
000138                  itsTypeStyle: TTypeStyle): TLegend;
000139          VAR itsString:      S255;
```

```
000140              itsLocation:    LPoint;
000141      BEGIN
000142          {$IFC fTrace}BP(11);{$ENDC}
000143          SetLPt(itsLocation, itsXLoc, itsYLoc); {=}
000144          NewStdLegend := TLegend.CREATE(NIL, heap, itsChars, itsView, itsLocation, itsTypeStyle);
000145          {$IFC fTrace}EP;{$ENDC}
000146      END;
000147
000148
000149
000150  METHODS OF TDialogWindow;
000151
000152
000153  {$S DlgAlloc}
000154  FUNCTION  TDialogWindow.CREATE(object: TObject; heap: THeap; itsResizability: BOOLEAN; itsHeight: INTEGER;
000155            itsKeyResponse, itsMenuResponse, itsDownInMainWindowResponse: TDiResponse): TDialogWindow;
000156      BEGIN
000157          {$IFC fTrace}BP(11);{$ENDC}
000158          IF object = NIL THEN
000159              object := NewObject(heap, THISCLASS);
000160          SELF := TDialogWindow(TDialogBox.CREATE(object, heap, itsResizability, itsHeight, itsKeyResponse,
000161                                                  itsMenuResponse, itsDownInMainWindowResponse));
000162
000163          SELF.controlPanel := SELF.selectPanel; {If not holding a TDialogView, client must reset}
000164          SELF.dialogView := NIL;
000165          SELF.mainDialog := NIL;
000166          {$IFC fTrace}EP;{$ENDC}
000167      END;
000168
000169
000170  {$IFC fDebugMethods}
000171  {$S DlgDbg}
000172  PROCEDURE TDialogWindow.Fields(PROCEDURE Field(nameAndType: S255));
000173  BEGIN
000174      SUPERSELF.Fields(Field);
000175      Field('controlPanel: TPanel');
000176      Field('dialogView: TDialogView');
000177      Field('mainDialog: TDialog');
000178      Field('');
000179  END;
000180  {$ENDC}
000181
000182
000183  {$S DlgHot}
000184  PROCEDURE TDialogWindow.Appear;
000185      PROCEDURE TellYourView(obj: TObject);
000186          PROCEDURE YouPrepare(obj: TObject);
000187              BEGIN
```

```
000188              TDialogImage(obj).PrepareToAppear;
000189              END;
000190      BEGIN
000191      IF InClass(TPanel(obj).view, TDialogView) THEN
000192          TDialogView(TPanel(obj).view).EachActualPart(YouPrepare);
000193      END;
000194  BEGIN
000195      {$IFC fTrace}BP(11);{$ENDC}
000196      SUPERSELF.Appear;
000197      SELF.panels.Each(TellYourView);
000198      {$IFC fTrace}EP;{$ENDC}
000199  END;
000200
000201
000202  {$S DlgHot}
000203  PROCEDURE TDialogWindow.BeDismissed;
000204      VAR dialogView:     TDialogView;
000205          defaultButton:  TButton;
000206  BEGIN
000207      {$IFC fTrace}BP(11);{$ENDC}
000208      PushFocus;
000209      SELF.Focus;
000210      IF InClass(SELF.controlPanel.view, TDialogView) THEN
000211          BEGIN
000212          dialogView := TDialogView(SELF.controlPanel.view);
000213          defaultButton := dialogView.defaultButton;
000214          IF defaultButton <> NIL THEN
000215              dialogView.PushButton(defaultButton)
000216              {may want to put in a delay loop here to assure hilit button seen}
000217
000218          ELSE {dialog box has no default button; just take it down}
000219              currentWindow.TakeDownDialogBox;
000220          END
000221      ELSE   {not a dialogView up there--must be a layout view}
000222          currentWindow.TakeDownDialogBox;
000223      PopFocus;
000224      {$IFC fTrace}EP;{$ENDC}
000225  END;
000226
000227
000228  {$S DlgHot}
000229  FUNCTION TDialogWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
000230  BEGIN
000231      {$IFC fTrace}BP(11);{$ENDC}
000232      CASE cmdNumber OF
000233          uEditDialog:
000234              CanDoCommand := TRUE;
000235          OTHERWISE
```

```
000236              CanDoCommand := currentWindow.CanDoCommand(cmdNumber, checkIt);
000237          END;
000238      {$IFC fTrace}EP;{$ENDC}
000239   END;
000240
000241
000242  {$S DlgHot}
000243  PROCEDURE TDialogWindow.Disappear;
000244  BEGIN
000245      {$IFC fTrace}BP(11);{$ENDC}
000246      SUPERSELF.Disappear;
000247      IF SELF.controlPanel <> NIL THEN
000248          IF InClass(SELF.controlPanel.view, TDialogView) THEN
000249              TDialogView(SELF.controlPanel.view).isShowing := FALSE;
000250      {$IFC fTrace}EP;{$ENDC}
000251  END;
000252
000253
000254  {$S DlgHot}
000255  FUNCTION  TDialogWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;
000256  BEGIN
000257      {$IFC fTrace}BP(12);{$ENDC}
000258      CASE cmdNumber OF
000259          uEditDialog:
000260              BEGIN
000261              IF SELF.controlPanel = NIL THEN
000262  {$IFC fDbgOK}
000263                  ABCBreak('DialogWindow.NewCommand NIL ctl pnl', 0)
000264  {$ENDC}
000265              ELSE
000266              IF NOT InClass(SELF.controlPanel.view, TDialogView) THEN
000267  {$IFC fDbgOK}
000268                  ABCBreak('DialogWindow.NewCommand, not a dialog view', 0)
000269  {$ENDC}
000270              ELSE
000271                  TDialogDesignWindow.CREATE(NIL, SELF.Heap, TDialogView(SELF.controlPanel.view)).SeizeControl;
000272              NewCommand := NIL;
000273              END;
000274          OTHERWISE
000275              NewCommand := currentWindow.NewCommand(cmdNumber);
000276          END;
000277      {$IFC fTrace}EP;{$ENDC}
000278  END;
000279
000280
000281  {$S DlgInit}
000282  BEGIN
000283      SetRect(stdPlainBorders, - stdLeftRightMargin, -stdSlimTitleHeight - stdBottomBorder, stdLeftRightMargin,
```

```
000284                  stdBottomBorder);
000285      SetRect(stdIDBorders , - stdLeftRightMargin, -stdTitleHeight - stdBottomBorder, stdLeftRightMargin,
000286                  stdBottomBorder);
000287      SetRect(stdThinBorders , 0, -stdSlimTitleHeight, 0, 0);
000288      SetPt(stdLabelOffset, 8, -2);
000289      SetPt(stdFrmeOffset, 20, 0);
000290      SetRect(stdFrameBorders, -30, -16, 30, 16);
000291      SetRect(stdHdngBorders, -6, -12, 6, 4);
000292
000293      MakeTypeStyle(famModern, size12Pitch, [], stdInputTypeStyle);
000294      MakeTypeStyle(famModern, size15Pitch, [], titleTypeStyle);
000295      MakeTypeStyle(famModern, size12Pitch, [], stdHdngTypeStyle);
000296
000297      copyright := 'Copyright 1983, 1984 by Apple Computer, Inc.';
000298
000299  END;
000300
000301
000302  METHODS OF TDialogView;
000303
000304
000305  {$S DlgAlloc}
000306  FUNCTION TDialogView.CREATE{(object: TObject; heap: THeap;  itsExtentLRect: LRect; itsPanel: TPanel;
000307                       itsPrintManager: TPrintManager; itsRes: Point)};
000308      VAR rootDialog:     TDialog;
000309          dialogWindow:   TDialogWindow;
000310          styleSheet:     TStyleSheet;
000311  BEGIN
000312      {$IFC fTrace}BP(11);{$ENDC}
000313
000314      IF object = NIL THEN
000315          object := NewObject(heap, THISCLASS);
000316      SELF := TDialogView(TView.CREATE(object, heap, itsPanel, itsExtentLRect, itsPrintManager, stdMargins,
000317                           (itsPrintManager <> NIL), itsRes, TRUE));
000318
000319      SELF.nonDialogExtent := itsExtentLRect;
000320
000321      rootDialog := TDialog.CREATE(NIL, heap, 'ROOT', SELF);
000322      SELF.rootDialog := rootDialog;  {create an empty master}
000323
000324      styleSheet := TStyleSheet.CREATE(NIL, heap);
000325      SELF.styleSheet := styleSheet;
000326
000327      WITH SELF DO
000328          BEGIN
000329          isShowing := FALSE; {not yet actually put up}
000330          currentDialogImage := NIL;
000331          defaultButton := NIL;
```

```
000332          hitButton := NIL;
000333
000334          paintFreeBoxes := FALSE;  {client can set this to TRUE after the CREATE call}
000335          paintSense := FALSE;
000336          startedPainting := FALSE;
000337
000338          mouseIsDown := FALSE;
000339          magnetCursor := noCursor;
000340          END;
000341
000342      IF InClass(itsPanel.window, TDialogWindow) THEN
000343          BEGIN
000344          dialogWindow := TDialogWindow(itsPanel.window);
000345          IF dialogWindow.controlPanel = NIL THEN
000346              dialogWindow.controlPanel := itsPanel;
000347          IF dialogWindow.dialogView = NIL THEN
000348              dialogWindow.dialogView := SELF;
000349          END;
000350     {$IFC fTrace}EP;{$ENDC}
000351  END;
000352
000353
000354  {$S DlgCold}
000355  PROCEDURE TDialogView.Free;
000356  BEGIN
000357      {$IFC fTrace}BP(11);{$ENDC}
000358      Free(SELF.rootDialog);
000359      Free(SELF.styleSheet);
000360      SUPERSELF.Free;
000361      {$IFC fTrace}EP;{$ENDC}
000362  END;
000363
000364
000365  {$IFC fDebugMethods}
000366  {$S DlgDbg}
000367  PROCEDURE TDialogView.Fields(PROCEDURE Field(nameAndType: S255));
000368  BEGIN
000369      SUPERSELF.Fields(Field);
000370      Field('rootDialog: TDialog');
000371      Field('nonDialogExtent: LRect');
000372      Field('currentDialogImage: TDialogImage');
000373      Field('defaultButton: TButton');
000374      Field('hitButton: TButton');
000375      Field('isShowing: BOOLEAN');
000376      Field('paintFreeBoxes: BOOLEAN');
000377      Field('paintSense: BOOLEAN');
000378      Field('startedPainting: BOOLEAN');
000379      Field('styleSheet: TStyleSheet');
```

```
000380      Field('mouseIsDown: BOOLEAN');
000381      Field('magnetCursor: INTEGER');
000382      Field('');
000383  END;
000384  {$ENDC}
000385
000386
000387  {$S DlgHot}
000388  PROCEDURE TDialogView.AbandonThatButton;
000389      PROCEDURE TurnOutTheLights;
000390          BEGIN
000391          SELF.hitButton.Highlight(hOnToOff);
000392          END;
000393  BEGIN
000394      {$IFC fTrace}BP(11);{$ENDC}
000395      IF SELF.hitButton <> NIL THEN
000396          BEGIN
000397          SELF.panel.OnAllPadsDo(TurnOutTheLights);
000398          IF SELF.currentDialogImage = SELF.hitButton THEN
000399              SELF.currentDialogImage := NIL;
000400          SELF.hitButton := NIL;
000401          END;
000402      {$IFC fTrace}EP;{$ENDC}
000403  END;
000404
000405
000406  {$S DlgAlloc}
000407  PROCEDURE TDialogView.AddDialog(dialog: TDialog);
000408      VAR dialogWindow:   TDialogWindow;
000409  BEGIN
000410      {$IFC fTrace}BP(11);{$ENDC}
000411      SELF.rootDialog.AddImage(dialog);
000412      IF InClass(SELF.panel.window, TDialogWindow) THEN
000413          BEGIN
000414          dialogWindow := TDialogWindow(SELF.panel.window);
000415          IF dialogWindow.mainDialog = NIL THEN
000416              dialogWindow.mainDialog := dialog;
000417          END;
000418      {$IFC fTrace}EP;{$ENDC}
000419  END;
000420
000421
000422  {$S DlgAlloc}
000423  FUNCTION TDialogView.AddNewDialog(itsKey: S4): TDialog;
000424      VAR dialogWindow:   TDialogWindow;
000425          dialog:         TDialog;
000426  BEGIN
000427      {$IFC fTrace}BP(11);{$ENDC}
```

```
000428      dialog := TDialog.CREATE(NIL, SELF.Heap, itsKey, SELF);
000429      SELF.AddDialog(dialog);
000430      AddNewDialog := dialog;
000431      {$IFC fTrace}EP;{$ENDC}
000432  END;
000433
000434
000435  {$S DlgAlloc}
000436  PROCEDURE TDialogView.ActivateDialog(dialog: TDialog; whichWay: BOOLEAN);
000437  BEGIN
000438      {$IFC fTrace}BP(11);{$ENDC}
000439      SELF.rootDialog.ActivateImage(dialog, whichWay);
000440      {$IFC fTrace}EP;{$ENDC}
000441  END;
000442
000443
000444  {$S DlgHot}
000445  PROCEDURE TDialogView.ButtonPushed(button: TButton);
000446      CONST   delayTime = 50000;
000447  VAR dialogView: TDialogView;
000448      command:    TCommand;
000449      sink:       LONGINT;
000450      i:          LONGINT;
000451  BEGIN
000452      {$IFC fTrace}BP(11);{$ENDC}
000453      IF currentWindow.dialogBox <> NIL THEN
000454          BEGIN
000455          dialogView := TDialogView(TDialogWindow(currentWindow.dialogBox).controlPanel.view);
000456          command := NIL;
000457          IF dialogView = SELF THEN
000458              BEGIN
000459              sink := 124395;
000460              FOR i := 1 TO delayTime DO
000461                  sink := sink - sink;
000462              dialogView.AbandonThatButton; {turn off highlighting just in case the dialog will be reused}
000463              IF button.cmdNumber <> noCmdNumber THEN
000464                  command := currentWindow.selectPanel.selection.NewCommand(button.cmdNumber);
000465              currentWindow.TakeDownDialogBox;
000466              IF command <> NIL THEN
000467                  currentWindow.PerformCommand(command);
000468              END;
000469          END;
000470      {$IFC fTrace}EP;{$ENDC}
000471  END;
000472
000473
000474  {$S DlgHot}
000475  PROCEDURE TDialogView.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN);
```

```
000476  {The client will occasionally want to override this, in order to change the display as an
000477   immediate consequence of a Checkbox being toggled}
000478      BEGIN
000479         {$IFC fTrace}BP(11);{$ENDC}
000480         { ... }
000481         {$IFC fTrace}EP;{$ENDC}
000482      END;
000483
000484
000485  {$S DlgHot}
000486  FUNCTION TDialogView.CursorAt(mouseLPt: LPoint): TCursorNumber;
000487      VAR cursorNumber:   TCursorNumber;
000488  BEGIN
000489     {$IFC fTrace}BP(11);{$ENDC}
000490      cursorNumber := noCursor;
000491      IF SELF.mouseIsDown AND (SELF.magnetCursor <> noCursor) THEN
000492          cursorNumber := SELF.magnetCursor
000493      ELSE
000494          BEGIN
000495          IF LRectHasLPt(SELF.rootDialog.extentLRect, mouseLPt) THEN
000496             cursorNumber := SELF.rootDialog.CursorAt(mouseLPt);
000497          IF cursorNumber = noCursor THEN
000498              cursorNumber := SELF.XCursorAt(mouseLPt);
000499          END;
000500
000501     CursorAt := cursorNumber;
000502     {$IFC fTrace}EP;{$ENDC}
000503  END;
000504
000505
000506  {$S DlgHot}
000507  PROCEDURE TDialogView.Draw;
000508      VAR s:             TListScanner;
000509          dialogImage:   TDialogImage;
000510  BEGIN
000511      {$IFC fTrace}BP(11);{$ENDC}
000512      SELF.isShowing := TRUE;  {update event triggered this}
000513      SELF.rootDialog.Draw;  {draw dialogs}
000514      SELF.XDraw;            {draw other stuff}
000515      {$IFC fTrace}EP;{$ENDC}
000516  END;
000517
000518
000519  {$S DlgHot}
000520  PROCEDURE TDialogView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
000521  BEGIN
000522      {$IFC fTrace}BP(11);{$ENDC}
000523      SELF.rootDialogImage.EachActualPart(doToObject);
```

```
000524      {$IFC fTrace}EP;{$ENDC}
000525  END;
000526
000527
000528  {$S DlgHot}
000529  PROCEDURE TDialogView.MouseMove(mouseLPt: LPoint);
000530      VAR currentDialogImage: TDialogImage;
000531  BEGIN
000532      {$IFC fTrace}BP(11);{$ENDC}
000533      currentDialogImage := SELF.currentDialogImage;
000534      IF currentDialogImage <> NIL THEN
000535          IF NOT currentDialogImage.StillMyMouse(mouseLPt) THEN
000536             currentDialogImage := NIL;
000537
000538      IF currentDialogImage = NIL THEN
000539          BEGIN
000540          currentDialogImage := SELF.rootDialog.DownAt(mouseLPt);
000541          IF currentDialogImage = NIL THEN
000542              SELF.XMouseMove(mouseLPt);
000543          END;
000544
000545      SELF.currentDialogImage := currentDialogImage;
000546     {$IFC fTrace}EP;{$ENDC}
000547  END;
000548
000549
000550  {$S DlgHot}
000551  PROCEDURE TDialogView.MousePress(mouseLPt: LPoint);
000552      VAR panel:              TPanel;
000553          takenBySelection:   BOOLEAN;
000554          currentDialogImage: TDialogImage;
000555  BEGIN
000556      {$IFC fTrace}BP(11);{$ENDC}
000557      panel := SELF.panel;
000558      SELF.startedPainting := FALSE;
000559      takenBySelection := FALSE;
000560
000561      SELF.mouseIsDown := TRUE;
000562      SELF.magnetCursor := noCursor;
000563
000564      currentDialogImage := SELF.currentDialogImage;
000565
000566      IF (currentDialogImage <> NIL) AND (SELF.panel.selection.kind <> nothingKind) THEN
000567          IF currentDialogImage.Hit(mouseLPt) THEN
000568              BEGIN
000569              SELF.panel.selection.MousePress(mouseLPt);
000570              takenBySelection := TRUE;
000571              END;
```

```
000572
000573      IF NOT takenBySelection THEN
000574          BEGIN
000575          panel.BeginSelection;
000576          currentDialogImage := SELF.rootDialog.DownAt(mouseLPt);
000577          IF currentDialogImage = NIL THEN
000578              SELF.XMousePress(mouseLPt)
000579          END;
000580
000581      SELF.currentDialogImage := currentDialogImage;
000582     {$IFC fTrace}EP;{$ENDC}
000583  END;
000584
000585
000586  {$S DlgHot}
000587  PROCEDURE TDialogView.MouseRelease;
000588  BEGIN
000589      {$IFC fTrace}BP(11);{$ENDC}
000590      SELF.mouseIsDown := FALSE;
000591      SELF.magnetCursor := noCursor;
000592      IF SELF.currentDialogImage <> NIL THEN
000593          SELF.currentDialogImage.MouseRelease
000594      ELSE
000595          SELF.XMouseRelease;
000596      {$IFC fTrace}EP;{$ENDC}
000597  END;
000598
000599  {$S DlgHot}
000600
000601  PROCEDURE TDialogView.PushButton(button: TButton);          {client or ToolKit may call}
000602  BEGIN
000603      {$IFC fTrace}BP(11);{$ENDC}
000604      IF InClass(button.parent, TDialog) THEN
000605          TDialog(button.parent).PushButton(button);
000606      {$IFC fTrace}EP;{$ENDC}
000607  END;
000608
000609
000610  {$S DlgAlloc}
000611  PROCEDURE TDialogView.RecalcExtent;
000612      VAR newExtent:  LRect;
000613  BEGIN
000614      {$IFC fTrace}BP(11);{$ENDC}
000615      IF NOT EmptyLRect(SELF.rootDialog.extentLRect) THEN
000616          BEGIN
000617          IF NOT EmptyLRect(SELF.nonDialogExtent) THEN
000618              UnionLRect(SELF.rootDialog.extentLRect, SELF.nonDialogExtent, newExtent);
000619          SELF.Resize(newExtent);
```

```
000620          END;
000621      {$IFC fTrace}EP;{$ENDC}
000622  END;
000623
000624
000625  {$S DlgAlloc}
000626  PROCEDURE TDialogView.RemoveDialog(dialog: TDialog; andFree: BOOLEAN);
000627  BEGIN
000628      {$IFC fTrace}BP(11);{$ENDC}
000629      SELF.rootDialog.DeleteImage(dialog, andFree);
000630      {$IFC fTrace}EP;{$ENDC}
000631  END;
000632
000633
000634  {$S DlgAlloc}
000635  PROCEDURE TDialogView.ReplaceDialog(oldDialog, newDialog: TDialog);
000636  BEGIN
000637      {$IFC fTrace}BP(11);{$ENDC}
000638      SELF.rootDialog.ReplaceImage(oldDialog, newDialog);
000639      {$IFC fTrace}EP;{$ENDC}
000640  END;
000641
000642
000643  {$S DlgAlloc}
000644  PROCEDURE TDialogView.SetDefaultButton(button: TButton);
000645      VAR thickPnSize:   point;
000646  BEGIN
000647      {$IFC fTrace}BP(11);{$ENDC}
000648      SELF.defaultButton := button;
000649      SetPt(thickPnSize, 3, 2);
000650      IF button <> NIL THEN
000651          button.buttonMetrics.penState.pnSize := thickPnSize;
000652      {$IFC fTrace}EP;{$ENDC}
000653  END;
000654
000655
000656  {$S DlgHot}
000657  FUNCTION TDialogView.XCursorAt(mouseLPt: LPoint): TCursorNumber;
000658  BEGIN
000659     {$IFC fTrace}BP(11);{$ENDC}
000660     XCursorAt := arrowCursor;
000661     {$IFC fTrace}EP;{$ENDC}
000662  END;
000663
000664
000665  {$S DlgHot}
000666  PROCEDURE TDialogView.XDraw;
000667  BEGIN
```

```
000668     {$IFC fTrace}BP(11);{$ENDC}
000669     {$IFC fTrace}EP;{$ENDC}
000670  END;
000671
000672
000673  {$S DlgHot}
000674  PROCEDURE TDialogView.XMousePress(mouseLPt: LPoint);
000675  BEGIN
000676     {$IFC fTrace}BP(11);{$ENDC}
000677     {$IFC fTrace}EP;{$ENDC}
000678  END;
000679
000680
000681  {$S DlgHot}
000682  PROCEDURE TDialogView.XMouseMove(mouseLPt: LPoint);
000683  BEGIN
000684     {$IFC fTrace}BP(11);{$ENDC}
000685     {$IFC fTrace}EP;{$ENDC}
000686  END;
000687
000688
000689  {$S DlgHot}
000690  PROCEDURE TDialogView.XMouseRelease;
000691  BEGIN
000692     {$IFC fTrace}BP(11);{$ENDC}
000693     {$IFC fTrace}EP;{$ENDC}
000694  END;
000695
000696
000697  {$S DlgInit}
000698  END;
000699
000700  {-------------------------------------------------------------------------------------------------------}
000701
000702
000703  METHODS OF TDialogImage;
000704
000705
000706  {$S TK2Start}
000707  FUNCTION  TDialogImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000708                 itsView: TView; withChildren: BOOLEAN): TDialogImage;
000709     VAR newList:    TList;
000710  BEGIN
000711     {$IFC fTrace}BP(11);{$ENDC}
000712     IF object = NIL THEN
000713         object := NewObject(heap, THISCLASS);
000714     SELF := TDialogImage(TImage.CREATE(object, heap, itsExtent, itsView));
000715
```

```
000716      WITH SELF DO
000717          BEGIN
000718          parent := NIL;
000719          isActive := TRUE;
000720          isEditable := TRUE;
000721          withID := FALSE;
000722          END;
000723      {$IFC fTrace}EP;{$ENDC}
000724  END;
000725
000726
000727  {$IFC fDebugMethods}
000728  {$S DlgDbg}
000729  PROCEDURE TDialogImage.Fields(PROCEDURE Field(nameAndType: S255));
000730  BEGIN
000731      SUPERSELF.Fields(Field);
000732      Field('parent: TDialogImage');
000733      Field('isActive: BOOLEAN');
000734      Field('isEditable: BOOLEAN');
000735      Field('withID: BOOLEAN');
000736      Field('');
000737  END;
000738  {$ENDC}
000739
000740
000741  {$S DlgCold}
000742  PROCEDURE TDialogImage.AddImage(dialogImage: TDialogImage);
000743  BEGIN
000744      {$IFC fTrace}BP(11);{$ENDC}
000745      {$IFC fTrace}EP;{$ENDC}
000746  END;
000747
000748
000749  {$S DlgAlloc}
000750  PROCEDURE TDialogImage.ActivateImage(dialogImage: TDialogImage; whichWay: BOOLEAN);
000751  BEGIN
000752      {$IFC fTrace}BP(11);{$ENDC}
000753      IF dialogImage.isActive <> whichWay THEN {state needs to change}
000754          BEGIN
000755          dialogImage.isActive := whichWay;
000756          SELF.view.panel.InvalLRect(dialogImage.extentLRect);  {??? Want to recalc my extent here???}
000757          END;
000758      {$IFC fTrace}EP;{$ENDC}
000759  END;
000760
000761
000762  {$S DlgCold}
000763  PROCEDURE TDialogImage.BringToFront(dialogImage: TDialogImage);
```

```
000764  BEGIN
000765      {$IFC fTrace}BP(11);{$ENDC}
000766      {$IFC fTrace}EP;{$ENDC}
000767  END;
000768
000769
000770  {$S DlgCold}
000771  PROCEDURE TDialogImage.ComeForward;
000772  BEGIN
000773      {$IFC fTrace}BP(11);{$ENDC}
000774      IF SELF.parent <> NIL THEN
000775          SELF.parent.BringToFront(SELF);
000776      {$IFC fTrace}EP;{$ENDC}
000777  END;
000778
000779
000780  {$S DlgHot}
000781  PROCEDURE TDialogImage.ControlHit(control: TDialogImage; toggleDirection: BOOLEAN);
000782  BEGIN
000783      {$IFC fTrace}BP(11);{$ENDC}
000784      IF SELF.parent <> NIL THEN
000785          SELF.parent.ControlHit(control, toggleDirection);
000786     {$IFC fTrace}EP;{$ENDC}
000787  END;
000788
000789
000790  {$S DlgAlloc}
000791  PROCEDURE TDialogImage.DeleteImage(dialogImage: TDialogImage; andFree: BOOLEAN);
000792  BEGIN
000793      {$IFC fTrace}BP(11);{$ENDC}
000794      {$IFC fTrace}EP;{$ENDC}
000795   END;
000796
000797
000798  {$S DlgHot}
000799  FUNCTION  TDialogImage.DownAt(mouseLPt: LPoint):  TDialogImage;
000800  BEGIN
000801      {$IFC fTrace}BP(11);{$ENDC}
000802      IF SELF.Hit(mouseLPt) THEN
000803          BEGIN
000804          SELF.MousePress(mouseLPt);
000805          DownAt := SELF;
000806          END
000807      ELSE
000808          DownAt := NIL;
000809      {$IFC fTrace}EP;{$ENDC}
000810  END;
000811
```

```
000812
000813  {$S DlgHot}
000814  PROCEDURE TDialogImage.Draw;
000815  BEGIN
000816      {$IFC fTrace}BP(11);{$ENDC}
000817      SELF.DrawJustMe;
000818      {$IFC fTrace}EP;{$ENDC}
000819  END;
000820
000821
000822  {$S DlgDbg} {by design}
000823  PROCEDURE TDialogImage.DrawJustMe;
000824  BEGIN
000825      {$IFC fTrace}BP(113);{$ENDC}
000826      {$IFC fTrace}EP;{$ENDC}
000827  END;
000828
000829
000830  {$S DlgHot}
000831  PROCEDURE TDialogImage.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
000832  BEGIN
000833      {$IFC fTrace}BP(11);{$ENDC}
000834      {overrides TImage's version, does specifically Nothing; TImageWithID redefines}
000835      {$IFC fTrace}EP;{$ENDC}
000836  END;
000837
000838
000839  {$S DlgHot}
000840  FUNCTION TDialogImage.HasId(id: S255): BOOLEAN;
000841  BEGIN
000842      {$IFC fTrace}BP(11);{$ENDC}
000843      HasID := FALSE;
000844      {$IFC fTrace}EP;{$ENDC}
000845  END;
000846
000847
000848  {$S SgLayout}
000849  FUNCTION TDialogImage.LaunchLayoutBox(view: TView): TImage;
000850      VAR myLayoutBox:    TLayoutBox;
000851          plannerView:    TPlannerView;
000852  BEGIN
000853      {$IFC fTrace}BP(10);{$ENDC}
000854      IF NOT SELF.isActive THEN
000855          LaunchLayoutBox := NIL
000856      ELSE
000857          LaunchLayoutBox :=  TLayoutBox.CREATE(NIL, view.Heap, SELF.extentLRect, '', NIL,
000858              view, SELF, stdPlainBorders, FALSE, FALSE, FALSE);
000859      {$IFC fTrace}EP;{$ENDC}
```

```
000860  END;
000861
000862
000863  {$S DlgHot}
000864  FUNCTION  TDialogImage.ObjectWithIDNumber(idNumber: INTEGER):  TDialogImage;
000865  BEGIN
000866      {$IFC fTrace}BP(11);{$ENDC}
000867      ObjectWithIDNumber := NIL;
000868      {$IFC fTrace}EP;{$ENDC}
000869  END;
000870
000871
000872  {$S DlgHot}
000873  FUNCTION  TDialogImage.ObjWithId(id: S255):  TDialogImage;
000874  BEGIN
000875      {$IFC fTrace}BP(11);{$ENDC}
000876      ObjWithId := NIL;
000877      {$IFC fTrace}EP;{$ENDC}
000878  END;
000879
000880
000881  {$S DlgHot}
000882  PROCEDURE TDialogImage.PrepareToAppear;
000883  BEGIN
000884      {$IFC fTrace}BP(11);{$ENDC}
000885      {$IFC fTrace}EP;{$ENDC}
000886  END;
000887
000888
000889  {$S DlgHot}
000890  PROCEDURE TDialogImage.RecalcExtent;
000891  BEGIN
000892      {$IFC fTrace}BP(11);{$ENDC}
000893      IF SELF.parent <> NIL THEN
000894          SELF.parent.RecalcExtent;
000895      {$IFC fTrace}EP;{$ENDC}
000896  END;
000897
000898  {$S DlgCold}
000899  PROCEDURE TDialogImage.ReplaceImage(replacee, newValue: TDialogImage);
000900  BEGIN
000901      {$IFC fTrace}BP(11);{$ENDC}
000902      {$IFC fTrace}EP;{$ENDC}
000903  END;
000904
000905  {$S DlgHot}
000906  FUNCTION  TDialogImage.StillMyMouse(mouseLPt: LPoint): BOOLEAN;
000907  BEGIN
```

```
000908      {$IFC fTrace}BP(11);{$ENDC}
000909      IF SELF.Hit(mouseLPt) THEN
000910          StillMyMouse := TRUE {I've handled it}
000911      ELSE
000912          StillMyMouse := FALSE; {give it to someone else}
000913     {default; this will usually be overridden in subclass}
000914      {$IFC fTrace}EP;{$ENDC}
000915  END;
000916
000917
000918  {$S DlgInit}
000919  END;
000920
000921
000922
000923  METHODS OF TImageWithID;
000924
000925  {$S DlgHot}
000926  FUNCTION  TImageWithID.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000927                  itsView: TView; withChildren: BOOLEAN): TImageWithID;
000928      VAR newList:        TList;
000929          newID:          TId;
000930          cState:         TConvResult;
000931          newIDNumber:    INTEGER;
000932  BEGIN
000933      {$IFC fTrace}BP(11);{$ENDC}
000934      IF object = NIL THEN
000935          object := NewObject(heap, THISCLASS);
000936      SELF := TImageWithID(TDialogImage.CREATE(object, heap, itsExtent, itsID, itsView, withChildren));
000937
000938      newID :=  Copy(itsId, 1, MIN(idLength, LENGTH(itsId)));
000939      StrUpperCased(@newID);
000940      StrToInt(@newID, newIDNumber, cState);
000941      IF cState <> cvValid THEN
000942          newIDNumber := noIDNumber;
000943
000944      WITH SELF DO
000945          BEGIN
000946          id :=  newID;
000947          idNumber := newIDNumber;
000948          withId := TRUE;
000949          END;
000950      IF withChildren THEN
000951          BEGIN
000952          newList := TList.CREATE(NIL, heap, 0);
000953          SELF.children := newList;
000954          END
000955      ELSE
```

```
000956          SELF.children := NIL;
000957      {$IFC fTrace}EP;{$ENDC}
000958  END;
000959
000960
000961  {$S DlgWarm}
000962  FUNCTION TImageWithID.Clone(heap: THeap): TObject;
000963      VAR children:       TList;
000964          copyOfMyself:   TImageWithID;
000965  BEGIN
000966      {$IFC fTrace}BP(11);{$ENDC}
000967      copyOfMyself := TImageWithID(SUPERSELF.Clone(heap));
000968      IF SELF.children <> NIL THEN
000969          BEGIN
000970          children := TList(SELF.children.Clone(heap));
000971          copyOfMyself.children := children;
000972          END;
000973      Clone := copyOfMyself;
000974      {$IFC fTrace}EP;{$ENDC}
000975  END;
000976
000977
000978  {$S DlgWarm}
000979  PROCEDURE TImageWithID.Free;
000980  BEGIN
000981      {$IFC fTrace}BP(11);{$ENDC}
000982      Free(SELF.children);
000983      SUPERSELF.Free;
000984      {$IFC fTrace}EP;{$ENDC}
000985  END;
000986
000987
000988  {$IFC fDebugMethods}
000989  {$S DlgDbg}
000990  PROCEDURE TImageWithID.Fields(PROCEDURE Field(nameAndType: S255));
000991  BEGIN
000992      SUPERSELF.Fields(Field);
000993      Field('children: TList');
000994      Field('id: STRING[9]');
000995      Field('idNumber: INTEGER');
000996      Field('');
000997  END;
000998  {$ENDC}
000999
001000
001001  {$S DlgHot}
001002  PROCEDURE TImageWithID.ActivateImage(dialogImage: TDialogImage; whichWay: BOOLEAN);
001003  BEGIN
```

```
001004      {$IFC fTrace}BP(11);{$ENDC}
001005      IF dialogImage.isActive <> whichWay THEN {state needs to change}
001006          BEGIN
001007          dialogImage.isActive := whichWay;
001008          SELF.view.panel.InvalLRect(dialogImage.extentLRect);
001009          END;
001010      {$IFC fTrace}EP;{$ENDC}
001011  END;
001012
001013
001014  {$S DlgHot}
001015  PROCEDURE TImageWithID.AddImage(dialogImage: TDialogImage);
001016  BEGIN
001017      {$IFC fTrace}BP(11);{$ENDC}
001018      SELF.children.InsLast(dialogImage);
001019      dialogImage.parent := SELF;
001020      dialogImage.HaveView(SELF.view);
001021      IF EmptyLRect(SELF.extentLRect) THEN
001022          SELF.extentLRect := dialogImage.extentLRect
001023      ELSE
001024  {$H-}   UnionLRect(SELF.extentLRect, dialogImage.extentLRect, SELF.extentLRect); {$H+}
001025      IF SELF.parent <> NIL THEN
001026          SELF.parent.RecalcExtent;
001027      {$IFC fTrace}EP;{$ENDC}
001028  END;
001029
001030
001031  {$S DlgHot}
001032  PROCEDURE TImageWithID.BringToFront(dialogImage: TDialogImage);
001033      VAR s:      TListScanner;
001034          image:  TDialogImage;
001035  BEGIN
001036      {$IFC fTrace}BP(11);{$ENDC}
001037      IF SELF.children <> NIL THEN
001038          BEGIN
001039          s := SELF.children.Scanner;
001040          WHILE s.Scan(image) DO
001041              IF image = dialogImage THEN
001042                  BEGIN
001043                  s.Delete(FALSE);
001044                  s.Done;
001045                  SELF.children.insLast(dialogImage);
001046                  END;
001047          END;
001048      IF SELF.parent <> NIL THEN
001049          SELF.parent.BringToFront(SELF);
001050      {$IFC fTrace}EP;{$ENDC}
001051  END;
```

```
001052
001053
001054  {$S DlgHot}
001055  FUNCTION  TImageWithID.CursorAt(mouseLPt: LPoint): TCursorNumber;
001056      VAR s:              TListScanner;
001057          dialogImage:    TDialogImage;
001058          cursorNumber:   TCursorNumber;
001059  {default: just passes the request on to children until one sets it}
001060  BEGIN
001061      {$IFC fTrace}BP(11);{$ENDC}
001062      cursorNumber := noCursor;
001063      IF LRectHasLPt(SELF.extentLRect, mouseLPt) THEN
001064          IF SELF.children <> NIL THEN
001065              BEGIN
001066              s := SELF.children.Scanner;
001067              WHILE s.Scan(dialogImage) DO
001068                  IF dialogImage.isActive THEN
001069                      BEGIN
001070                      cursorNumber := dialogImage.CursorAt(mouseLPt);
001071                      IF cursorNumber <> noCursor THEN
001072                          s.Done;
001073                      END;
001074              END;
001075      CursorAt := cursorNumber;
001076      {$IFC fTrace}EP;{$ENDC}
001077  END;
001078
001079
001080  {$S DlgCold}
001081  PROCEDURE TImageWithID.DeleteImage(dialogImage: TDialogImage; andFree: BOOLEAN);
001082  {deletes the indicated dialogImage from my children}
001083      VAR s:              TListScanner;
001084          aDialogImage:   TDialogImage;
001085  BEGIN
001086      {$IFC fTrace}BP(11);{$ENDC}
001087      s:= SELF.children.Scanner;
001088      WHILE s.Scan(aDialogImage) DO
001089          IF aDialogImage = dialogImage THEN
001090              BEGIN
001091              IF (TDialogView(SELF.view).isShowing) AND (dialogImage.isActive) THEN
001092                  SELF.view.panel.InvalLRect(dialogImage.extentLRect);
001093              s.Delete(andFree);
001094              s.Done;
001095              END;
001096      {$IFC fTrace}EP;{$ENDC}
001097   END;
001098
001099
```

```
001100  {$S DlgHot}
001101  PROCEDURE TImageWithID.Draw;
001102      PROCEDURE YouDraw(obj: TObject);
001103          VAR dialogImage: TDialogImage;
001104      BEGIN
001105          dialogImage := TDialogImage(obj);
001106          IF dialogImage.isActive THEN
001107              dialogImage.Draw;
001108      END;
001109  BEGIN
001110      {$IFC fTrace}BP(11);{$ENDC}
001111      IF LRectIsVisible(SELF.extentLRect) THEN
001112          BEGIN
001113          SELF.EachActualPart(YouDraw);
001114          SELF.DrawJustMe;
001115          END;
001116      {$IFC fTrace}EP;{$ENDC}
001117  END;
001118
001119
001120  {$S DlgHot}
001121  PROCEDURE TImageWithID.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
001122  BEGIN
001123      {$IFC fTrace}BP(11);{$ENDC}
001124      IF SELF.children <> NIL THEN
001125          SELF.children.Each(DoToObject);
001126      {$IFC fTrace}EP;{$ENDC}
001127  END;
001128
001129
001130  {$S DlgHot}
001131  PROCEDURE TImageWithID.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
001132  BEGIN
001133      {$IFC fTrace}BP(11);{$ENDC}
001134      SELF.EachActualPart(DoToObject); {???}
001135     {$IFC fTrace}EP;{$ENDC}
001136  END;
001137
001138
001139  {$S DlgHot}
001140  FUNCTION TImageWithID.HasId(id: S255): BOOLEAN;
001141  BEGIN
001142      {$IFC fTrace}BP(11);{$ENDC}
001143  {$H-} id :=  Copy(id, 1, MIN(idLength, LENGTH(id))); {$H+}
001144      StrUpperCased(@id);
001145
001146      IF SELF.id = id THEN
001147          HasId := TRUE
```

```
001148      ELSE
001149          HasId := FALSE;
001150      {$IFC fTrace}EP;{$ENDC}
001151  END;
001152
001153
001154  {$S DlgHot}
001155  PROCEDURE  TImageWithID.HaveView(view: TView);
001156      PROCEDURE YouHaveView(obj: TObject);
001157          VAR dialogImage:  TDialogImage;
001158      BEGIN
001159          dialogImage := TDialogImage(obj);
001160          dialogImage.HaveView(view);
001161      END;
001162  BEGIN
001163      {$IFC fTrace}BP(11);{$ENDC}
001164      SELF.view := view;
001165      SELF.EachActualPart(YouHaveView);
001166      {$IFC fTrace}EP;{$ENDC}
001167  END;
001168
001169
001170  {$S SgLayout}
001171  FUNCTION TImageWithID.LaunchLayoutBox(view: TView): TImage;
001172      VAR dialogImage:        TDialogImage;
001173          newExtent:          LRect;
001174          boxTitle:           S255;
001175          theString:          TLegend;
001176          childsLayoutBox:    TLayoutBox;
001177          myLayoutBox:        TLayoutBox;
001178          plannerView:        TPlannerView;
001179          postChildExtent:    LRect;
001180          withChildren:       BOOLEAN;
001181      PROCEDURE YouMakeLayoutBoxes(obj: TObject);
001182              VAR dialogImage:    TDialogImage;
001183                  interimImage:   TImage;
001184          BEGIN
001185          dialogImage := TDialogImage(obj);
001186          interimImage := dialogImage.LaunchLayoutBox(view);
001187          IF interimImage <> NIL THEN
001188              BEGIN
001189              childsLayoutBox := TLayoutBox(interimImage);
001190              myLayoutBox.AddImage(childsLayoutBox);
001191              UnionLRect(postChildExtent, childsLayoutBox.extentLRect, postChildExtent);
001192              END;
001193          END;
001194  BEGIN
001195      {$IFC fTrace}BP(10);{$ENDC}
```

```
001196      IF NOT SELF.isActive THEN
001197         LaunchLayoutBox := NIL
001198      ELSE
001199          BEGIN {=}
001200          plannerView := TPlannerView(view);
001201          IF SELF.ID <> '' THEN
001202              boxTitle := SELF.id
001203          ELSE
001204          IF SELF.idNumber = noIDNumber THEN
001205              boxTitle := ''
001206          ELSE
001207              IntToString(SELF.idNumber, @boxTitle);
001208
001209          withChildren := (SELF.children <> NIL);
001210          IF withChildren THEN
001211              withChildren := SELF.children.Size > 0;
001212
001213          myLayoutBox := TLayoutBox.CREATE(NIL, SELF.Heap, SELF.extentLRect, boxTitle, NIL, plannerView, SELF,
001214                          stdIDBorders, FALSE, withChildren, withChildren);
001215                     {default for a dialogImage is for the layout box to SUPPRESS drawing the manipulee}
001216
001217          postChildExtent := SELF.extentLRect; {i.e., WITHOUT my borders}
001218          SELF.EachActualPart(YouMakeLayoutBoxes); {tells my children to make their own layout
001219                                                    boxes; may grow postChildExtent}
001220
001221          myLayoutBox.RecalcExtent;
001222          LaunchLayoutBox := myLayoutBox;
001223          END;
001224      {$IFC fTrace}EP;{$ENDC}
001225  END;
001226
001227
001228  {$S DlgHot}
001229  FUNCTION  TImageWithID.ObjectWithIDNumber(idNumber: INTEGER):  TDialogImage;
001230      VAR s:              TListScanner;
001231          dialogImage:    TDialogImage;
001232  BEGIN
001233      {$IFC fTrace}BP(11);{$ENDC}
001234      ObjectWithIDNumber := NIL;
001235      IF SELF.children <> NIL THEN
001236          BEGIN
001237          s := SELF.children.Scanner;
001238          WHILE s.Scan(dialogImage) DO
001239              IF dialogImage.withID THEN
001240                  IF TImageWithID(dialogImage).idNumber = idNumber THEN
001241                      BEGIN
001242                      ObjectWithIDNumber := dialogImage;
001243                      s.Done;
```

```
001244                      END;
001245          END;
001246      {$IFC fTrace}EP;{$ENDC}
001247  END;
001248
001249
001250  {$S DlgHot}
001251  FUNCTION  TImageWithID.ObjWithId(id: S255):  TDialogImage;
001252      VAR s:              TListScanner;
001253          dialogImage:    TDialogImage;
001254  BEGIN
001255      {$IFC fTrace}BP(11);{$ENDC}
001256      id :=  Copy(id, 1, MIN(idLength, LENGTH(id)));
001257      StrUpperCased(@id);
001258      ObjWithId := NIL;
001259      IF SELF.children <> NIL THEN
001260          BEGIN
001261          s := SELF.children.Scanner;
001262          WHILE s.Scan(dialogImage) DO
001263              IF dialogImage.withID THEN
001264                  IF TImageWithID(dialogImage).id = id THEN
001265                      BEGIN
001266                      ObjWithId := dialogImage;
001267                      s.Done;
001268                      END;
001269          END;
001270      {$IFC fTrace}EP;{$ENDC}
001271  END;
001272
001273
001274  {$S DlgHot}
001275  PROCEDURE TImageWithID.OffSetBy(deltaLPt: LPoint);
001276      PROCEDURE YouOffset(obj: TObject);
001277      BEGIN
001278          TDialogImage(obj).OffsetBy(deltaLPt);
001279      END;
001280  BEGIN
001281      {$IFC fTrace}BP(11);{$ENDC}
001282  {$H-} OffsetLRect(SELF.extentLRect, deltaLPt.h, deltaLPt.v); {$H+}
001283      SELF.EachActualPart(YouOffset);  {tells children}
001284      {$IFC fTrace}EP;{$ENDC}
001285  END;
001286
001287
001288  {$S DlgWarm}
001289  PROCEDURE TImageWithID.RecalcExtent;  {a bottom-up communication line; child who changes tells
001290                                         his parent, who changes himself and then tells HIS parent}
001291      VAR s:              TListScanner;
```

```
001292          dialogImage:    TDialogImage;
001293          newExtent:      LRect;
001294  BEGIN
001295      {$IFC fTrace}BP(11);{$ENDC}
001296      {can be speeded up}
001297      IF SELF.children <> NIL THEN
001298          IF SELF.children.Size > 0 THEN
001299              BEGIN
001300              newExtent := zeroLRect;
001301              s := SELF.children.Scanner;
001302              WHILE s.Scan(dialogImage) DO
001303                  IF EmptyLRect(newExtent) THEN
001304                      newExtent := dialogImage.extentLRect
001305                  ELSE
001306                      UnionLRect(newExtent, dialogImage.extentLRect, newExtent);
001307              SELF.Resize(newExtent);
001308              END;
001309      IF SELF.parent <> NIL THEN
001310          SELF.parent.RecalcExtent;
001311      {$IFC fTrace}EP;{$ENDC}
001312  END;
001313
001314
001315  {$S DlgCold}
001316  PROCEDURE TImageWithID.ReplaceImage(replacee, newValue: TDialogImage);
001317   {make this such that it puts back at same place; or use Become}
001318  BEGIN
001319      {$IFC fTrace}BP(11);{$ENDC}
001320      SELF.DeleteImage(replacee, TRUE);
001321      SELF.AddImage(newValue);
001322      {$IFC fTrace}EP;{$ENDC}
001323  END;
001324
001325
001326  {$S DlgHot}
001327  FUNCTION  TImageWithID.StillMyMouse(mouseLPt : LPoint): BOOLEAN;
001328  BEGIN
001329      {$IFC fTrace}BP(11);{$ENDC}
001330      IF SELF.Hit(mouseLPt) THEN
001331          StillMyMouse := TRUE {I've handled it}
001332      ELSE
001333          StillMyMouse := FALSE; {give it to someone else}
001334     {default; this will usually be overridden in subclass}
001335      {$IFC fTrace}EP;{$ENDC}
001336  END;
001337
001338
001339  {$S DlgInit}
```

```
001340  END;
001341
001342
001343  METHODS OF TDialog;
001344
001345
001346  {$S DlgAlloc}
001347  FUNCTION TDialog.CREATE(object: TObject; heap: THeap; itsKey: S4; itsView: TView): TDialog;
001348      VAR itsStringKey:   TStringKey;
001349          itsExtent:      LRect;
001350  BEGIN
001351      {$IFC fTrace}BP(11);{$ENDC}
001352      WITH stdButtonMetrics DO
001353          BEGIN
001354          height := stdBtnHeight;
001355          curvH := stdCurvH;
001356          curvV := stdCurvV;
001357          typeStyle := sysTypeStyle;
001358          penState := normalPen;
001359          expandNum := 4;
001360          expandDen := 3;
001361          absMinWidth := 80;
001362          END;
001363      IF object = NIL THEN
001364          object := NewObject(heap, THISCLASS);
001365      SELF := TDialog(TImageWithID.CREATE(object, heap, zeroLRect, itsKey, itsView, TRUE));
001366      XferLeft(Ptr(ORD(@itsKey)+1), @itsStringKey.trueKey, 4);  {get it into LONGINT form}
001367      itsStringKey.key := itsKey;
001368      SELF.stringKey := itsStringKey;
001369     {$IFC fTrace}EP;{$ENDC}
001370  END;
001371
001372
001373  {$IFC fDebugMethods}
001374  {$S DlgDbg}
001375  PROCEDURE TDialog.Fields(PROCEDURE Field(nameAndType: S255));
001376  BEGIN
001377      SUPERSELF.Fields(Field);
001378      Field('stringKey: RECORD trueKey: LONGINT; key: STRING[4] END');
001379      Field('');
001380  END;
001381  {$ENDC}
001382
001383
001384  {$S DlgAlloc}
001385  FUNCTION  TDialog.NewButton(itsPhrase: INTEGER; itsMetrics: TButtonMetrics; sameSizedButton: TButton;
001386                      itsCmdNumber: TCmdNumber): TButton;
001387      VAR itsID:          S255;
```

```
001388          itsLocation:    LPoint;
001389          button:         TButton;
001390  BEGIN
001391      {$IFC fTrace}BP(11);{$ENDC}
001392      GetTextAndLocation(itsPhrase, itsID, itsLocation);
001393      button := SELF.AddButton(itsID, itsLocation, itsMetrics, sameSizedButton, itsCmdNumber);
001394      button.idNumber := itsPhrase;
001395      NewButton := button;
001396      {$IFC fTrace}EP;{$ENDC}
001397  END;
001398
001399
001400  {$S DlgAlloc}
001401  FUNCTION  TDialog.NewCluster(itsPhrase: INTEGER): TCluster;
001402      VAR itsID:          S255;
001403          itsLocation:    LPoint;
001404          cluster:        TCluster;
001405  BEGIN
001406      {$IFC fTrace}BP(11);{$ENDC}
001407      GetTextAndLocation(itsPhrase, itsID, itsLocation);
001408      cluster := SELF.AddStdCluster(itsID, itsLocation.h, itsLocation.v);
001409      cluster.idNumber := itsPhrase;
001410      NewCluster := cluster;
001411      {$IFC fTrace}EP;{$ENDC}
001412  END;
001413
001414
001415  {$S DlgAlloc}
001416  FUNCTION  TDialog.NewFreeCheckbox(itsPhrase: INTEGER;  boxWidth: INTEGER;
001417      boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckBox;
001418      VAR itsID:          S255;
001419          itsLocation:    LPoint;
001420          checkbox:       TCheckbox;
001421  BEGIN
001422      {$IFC fTrace}BP(11);{$ENDC}
001423      GetTextAndLocation(itsPhrase, itsID, itsLocation);
001424      checkbox := SELF.AddFreeCheckbox(itsID, itsLocation.h, itsLocation.v, boxWidth, boxHeight,
001425          wantLabel, labelOffset, itsTypeStyle);
001426      checkbox.idNumber := itsPhrase;
001427      NewFreeCheckbox := checkbox;
001428      {$IFC fTrace}EP;{$ENDC}
001429  END;
001430
001431
001432  {$S DlgAlloc}
001433  FUNCTION  TDialog.NewInputFrame(itsPhrase: INTEGER; promptTypeStyle: TTypeStyle;
001434          inputOffset: Point; inputTypeStyle: TTypeStyle;
001435          maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect: BOOLEAN;
```

```
001436          drawHitLRect: BOOLEAN): TInputFrame;
001437      VAR itsID:          S255;
001438          itsLocation:    LPoint;
001439          inputFrame:     TInputFrame;
001440          inputLocation:  LPoint;
001441  BEGIN
001442      {$IFC fTrace}BP(11);{$ENDC}
001443      GetTextAndLocation(itsPhrase, itsID, itsLocation);
001444
001445      SetQDTypeStyle(promptTypeStyle);
001446      WITH inputLocation DO
001447          BEGIN
001448          h := itsLocation.h + StringWidth(itsID) + inputOffset.h;
001449          v := itsLocation.v + inputOffset.v;
001450          END;
001451
001452      inputFrame := SELF.AddInputFrame(itsID, itsLocation, promptTypeStyle,
001453          inputLocation, inputTypeStyle, maxInputChars, itsBorders, drawInputLRect,
001454          drawHitLRect);
001455      inputFrame.idNumber := itsPhrase;
001456      NewInputFrame := inputFrame;
001457      {$IFC fTrace}EP;{$ENDC}
001458  END;
001459
001460
001461  {$S DlgAlloc}
001462  FUNCTION  TDialog.NewLegend(itsPhrase: INTEGER; itsTypeStyle: TTypeStyle): TLegend;
001463      VAR itsID:          S255;
001464          itsLocation:    LPoint;
001465          legend:         TLegend;
001466  BEGIN
001467      {$IFC fTrace}BP(11);{$ENDC}
001468      GetTextAndLocation(itsPhrase, itsID, itsLocation);
001469      legend := SELF.AddStdLegend(itsID, itsLocation.h, itsLocation.v, itsTypeStyle);
001470      NewLegend := legend;
001471      {$IFC fTrace}EP;{$ENDC}
001472  END;
001473
001474
001475  {$S DlgAlloc}
001476  FUNCTION  TDialog.NewRowOfBoxes(itsPhrase: INTEGER; numberOfBoxes: INTEGER;
001477      startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER): TCluster;
001478      VAR itsID:          S255;
001479          itsLocation:    LPoint;
001480          cluster:        TCluster;
001481  BEGIN
001482      {$IFC fTrace}BP(11);{$ENDC}
001483      GetTextAndLocation(itsPhrase, itsID, itsLocation);
```

```
001484      cluster := SELF.AddRowOfBoxes(itsID, itsLocation.h, itsLocation.v, numberOfBoxes,
001485          startingIDNumber, boxWidth, boxHeight, boxSpacing);
001486      cluster.idNumber := itsPhrase;
001487      NewRowOfBoxes := cluster;
001488      {$IFC fTrace}EP;{$ENDC}
001489  END;
001490
001491
001492  {$S DlgAlloc}
001493  PROCEDURE TDialog.AddOKButton(cmdNumber: TCmdNumber);
001494      VAR button: TButton;
001495  BEGIN
001496      {$IFC fTrace}BP(11);{$ENDC}
001497      button := SELF.NewButton(phOK, stdButtonMetrics,
001498                  TButton(SELF.ObjectWithIDNumber(phCancel)), cmdNumber);
001499      {$IFC fTrace}EP;{$ENDC}
001500  END;
001501
001502
001503  {$S DlgAlloc}
001504  PROCEDURE TDialog.AddCancelButton(cmdNumber: TCmdNumber);
001505      VAR button: TButton;
001506  BEGIN
001507      {$IFC fTrace}BP(11);{$ENDC}
001508      button := SELF.NewButton(phCancel, stdButtonMetrics,
001509                  TButton(SELF.ObjectWithIDNumber(phOK)), cmdNumber);
001510      {$IFC fTrace}EP;{$ENDC}
001511  END;
001512
001513
001514  {$S DlgAlloc}
001515  FUNCTION  TDialog.AddBigFreeCheckbox(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCheckbox; {---}
001516      VAR location:       LPoint;
001517          itsChars:       S255;
001518          newBox:         TCheckbox;
001519          labelOffset:    Point;
001520          typeStyle:      TTypeStyle;
001521      BEGIN
001522          {$IFC fTrace}BP(11);{$ENDC}
001523          SetLPt(location, itsXLoc, itsYLoc);
001524          SetPt(labelOffset, 20, -4);
001525          MakeTypeStyle(famClassic, size18Point, [], typeStyle);
001526          newBox := TCheckbox.CREATE(NIL, SELF.Heap, itsId, SELF.view, location, 36, 24, TRUE,
001527              labelOffset, typeStyle);
001528          SELF.AddImage(newBox);
001529          AddBigFreeCheckbox := newBox;
001530          {$IFC fTrace}EP;{$ENDC}
001531      END;
```

```
001532
001533
001534  {$S DlgAlloc}
001535  FUNCTION  TDialog.AddButton(itsId: S255; itsLocation: LPoint; itsMetrics: TButtonMetrics;
001536          sameSizedButton: TButton; itsCmdNumber: TCmdNumber): TButton;
001537      VAR button: TButton;
001538  BEGIN
001539      {$IFC fTrace}BP(11);{$ENDC}
001540      button := TButton.CREATE(NIL, SELF.Heap, itsId, SELF.view, itsLocation, itsMetrics,
001541          sameSizedButton, itsCmdNumber);
001542      SELF.AddImage(button);
001543      AddButton := button;
001544      {$IFC fTrace}EP;{$ENDC}
001545  END;
001546
001547
001548  {$S DlgAlloc}
001549  FUNCTION  TDialog.AddStdButton(itsId: S255; itsXLoc, itsYLoc: LONGINT;  sameSizedButton: TButton;
001550                  itsCmdNumber: TCmdNumber): TButton;
001551  VAR location:   LPoint;
001552  BEGIN
001553      {$IFC fTrace}BP(11);{$ENDC}
001554      SetLPt(location, itsXLoc, itsYLoc);
001555      AddStdButton := SELF.AddButton(itsID, location, stdButtonMetrics, sameSizedButton, itsCmdNumber);
001556      {$IFC fTrace}EP;{$ENDC}
001557  END;
001558
001559
001560  {$S DlgAlloc}
001561  FUNCTION TDialog.AddStdFreeCheckbox(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCheckBox;
001562  VAR legend:     TLegend;
001563      location:   LPoint;
001564      checkbox:   TCheckbox;
001565  BEGIN
001566     {$IFC fTrace}BP(11);{$ENDC}
001567     SetLPt (location, itsXLoc, itsYLoc);
001568     checkbox := TCheckbox.CREATE(NIL, SELF.Heap, itsId, SELF.view, location, stdBoxWidth,
001569          stdBoxHeight, TRUE, stdLabelOffset, sysTypeStyle);
001570     SELF.AddImage(checkbox);
001571     AddStdFreeCheckbox := checkbox;
001572     {$IFC fTrace}EP;{$ENDC}
001573  END;
001574
001575
001576  {$S DlgAlloc}
001577  FUNCTION TDialog.AddStdCluster(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCluster;
001578      VAR location:   LPoint;
001579          cluster:    TCluster;
```

```
001580      BEGIN
001581          {$IFC fTrace}BP(11);{$ENDC}
001582          SetLPt(location, itsXLoc, itsYLoc);
001583          cluster :=TCluster.CREATE(NIL, SELF.Heap, itsId, SELF.view, location);
001584          SELF.AddImage(cluster);
001585          AddStdCluster := cluster;
001586          {$IFC fTrace}EP;{$ENDC}
001587      END;
001588
001589
001590  {$S DlgAlloc}
001591  FUNCTION  TDialog.AddFreeCheckbox(itsID: S255; itsXLoc, itsYLoc: LONGINT; boxWidth: INTEGER;
001592          boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox;
001593      VAR location:   LPoint;
001594          checkbox:   TCheckbox;
001595      BEGIN
001596          {$IFC fTrace}BP(11);{$ENDC}
001597          SetLPt(location, itsXLoc, itsYLoc);
001598          checkbox := TCheckbox.CREATE(NIL, SELF.Heap, itsID, SELF.view, location, boxWidth, boxHeight,
001599                          wantLabel, labelOffset, itsTypeStyle);
001600          SELF.AddImage(checkbox);
001601          AddFreeCheckbox := checkbox;
001602          {$IFC fTrace}EP;{$ENDC}
001603      END;
001604
001605
001606  {$S DlgAlloc}
001607  FUNCTION  TDialog.AddInputFrame(itsId: S255;
001608          promptLocation: LPoint; promptTypeStyle: TTypeStyle;
001609          inputLocation:  LPoint; inputTypeStyle: TTypeStyle;
001610          maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect: BOOLEAN;
001611          drawHitLRect: BOOLEAN): TInputFrame;
001612      VAR inputFrame: TInputFrame;
001613  BEGIN
001614     {$IFC fTrace}BP(11);{$ENDC}
001615     inputFrame := TInputFrame.CREATE(NIL, SELF.Heap, itsID, SELF.view, promptLocation, promptTypeStyle,
001616          inputLocation, inputTypeStyle, maxInputChars, itsBorders,
001617          drawInputLRect, drawHitLRect);
001618     SELF.AddImage(inputFrame);
001619     AddInputFrame := inputFrame;
001620     {$IFC fTrace}EP;{$ENDC}
001621   END;
001622
001623
001624  {$S DlgAlloc}
001625  FUNCTION TDialog.AddRowOfBoxes(itsID: S255; itsXLoc, itsYLoc: LONGINT; numberOfBoxes: INTEGER;
001626      startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER): TCluster;
001627      VAR currentIDNumber: INTEGER;
```

```
001628          checkbox:       TCheckbox;
001629          newLocation:    LPoint;
001630          newID:          S255;
001631          cluster:        TCluster;
001632  BEGIN
001633      {$IFC fTrace}BP(11);{$ENDC}
001634      cluster := SELF.AddStdCluster(itsID, itsXLoc, itsYLoc);
001635      FOR currentIDNumber := startingIDNumber TO (startingIDNumber + numberOfBoxes -  1) DO
001636          BEGIN
001637          IF cluster.lastBox = NIL THEN {this is the first to be inserted}
001638              newLocation := cluster.location
001639          ELSE { There is already at least one box in the cluster -- align to right of it}
001640              WITH newLocation DO
001641                  BEGIN
001642                  h := cluster.lastBox.rectImage.extentLRect.right + boxSpacing;  {??}
001643                  v := cluster.lastBox.rectImage.extentLRect.top;
001644                  END;
001645          IntToString(currentIDNumber, @newID);
001646          checkbox := cluster.AddCheckbox(newID, newLocation, boxWidth, boxHeight, FALSE, zeroPt,
001647              sysTypeStyle, FALSE);
001648          checkBox.IDNumber := currentIDNumber;
001649          END;
001650      AddRowOfBoxes := cluster;
001651      {$IFC fTrace}EP;{$ENDC}
001652  END;
001653
001654
001655  {$S DlgAlloc}
001656  FUNCTION  TDialog.AddStdInputFrame(itsId: S255; itsXLoc: LONGINT;
001657          itsYLoc: LONGINT; maxInputChars : INTEGER): TInputFrame;
001658      VAR promptLocation: LPoint;
001659          inputLocation:  LPoint;
001660          inputFrame:     TInputFrame;
001661  BEGIN
001662     {$IFC fTrace}BP(11);{$ENDC}
001663      SetLPt(promptLocation, itsXLoc, itsYLoc);
001664
001665      SetQDTypeStyle(sysTypeStyle);
001666
001667      WITH inputLocation DO
001668          BEGIN
001669          h := itsXLoc + StringWidth(itsID) + 20;
001670          v := itsYLoc;
001671          END;
001672
001673      inputFrame := TInputFrame.CREATE(NIL, SELF.Heap, itsId, SELF.view, promptLocation, sysTypeStyle,
001674          inputLocation, stdInputTypeStyle, maxInputChars,
001675          stdFrameBorders, TRUE {draw input LRect}, TRUE {draw HitLRect});
```

```
001676      SELF.AddImage(inputFrame);
001677      AddStdInputFrame := inputFrame;
001678      {$IFC fTrace}EP;{$ENDC}
001679    END;
001680
001681
001682  {$S DlgAlloc}
001683  FUNCTION  TDialog.AddStdLegend(itsId: S255; itsXLoc, itsYLoc: LONGINT;
001684          itsTypeStyle: TTypeStyle): TLegend;
001685      VAR newString:  TLegend;
001686  BEGIN
001687      {$IFC fTrace}BP(11);{$ENDC}
001688      newString := NewStdLegend(SELF.Heap, itsID, itsXLoc, itsYLoc, SELF.view, itsTypeStyle);
001689      SELF.AddImage(newString);
001690      AddStdLegend := newString;
001691      {$IFC fTrace}EP;{$ENDC}
001692  END;
001693
001694
001695  {$S DlgAlloc}
001696  FUNCTION  TDialog.AddSysLegend(itsId: S255; itsXLoc, itsYLoc: LONGINT): TLegend;
001697      VAR newString:  TLegend;
001698  BEGIN
001699      {$IFC fTrace}BP(11);{$ENDC}
001700      newString := NewSysLegend(SELF.Heap, itsID, itsXLoc, itsYLoc, SELF.view);
001701      SELF.AddImage(newString);
001702      AddSysLegend := newString;
001703      {$IFC fTrace}EP;{$ENDC}
001704  END;
001705
001706
001707  {$S DlgHot}
001708  PROCEDURE TDialog.ButtonPushed(button: TButton);  {usually'll be called through SUPERSELF}
001709  BEGIN
001710      {$IFC fTrace}BP(11);{$ENDC}
001711      IF SELF.parent <> NIL THEN
001712          TDialog(SELF.parent).ButtonPushed(button)
001713      ELSE
001714          TDialogView(SELF.view).ButtonPushed(button);
001715      {$IFC fTrace}EP;{$ENDC}
001716  END;
001717
001718
001719  {$S DlgHot}
001720  PROCEDURE TDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN);
001721      {default--passes up the line; client can reimplement}
001722  BEGIN
001723      {$IFC fTrace}BP(11);{$ENDC}
```

```
001724      IF SELF.parent <> NIL THEN
001725          BEGIN
001726          IF InClass(SELF.parent, TDialog) THEN
001727              TDialog(SELF.parent).CheckboxHit(checkbox, toggleDirection)
001728          ELSE
001729              SELF.parent.ControlHit(checkbox, toggleDirection);
001730          END
001731      ELSE
001732          TDialogView(SELF.view).CheckboxHit(checkbox, toggleDirection);
001733      {$IFC fTrace}EP;{$ENDC}
001734  END;
001735
001736
001737  {$S DlgHot}
001738  PROCEDURE TDialog.ControlHit(control: TDialogImage; toggleDirection: BOOLEAN);
001739  BEGIN
001740      {$IFC fTrace}BP(11);{$ENDC}
001741      IF InClass(control, TButton) THEN
001742          SELF.ButtonPushed(TButton(control))  {this branch perhaps not achievable in current design}
001743      ELSE
001744      IF InClass(control, TCheckbox) THEN
001745          SELF.CheckboxHit(TCheckbox(control), toggleDirection);   {Client can add own kinds by redefining this}
001746     {$IFC fTrace}EP;{$ENDC}
001747  END;
001748
001749
001750  {$S DlgHot}
001751  FUNCTION TDialog.DownAt(mouseLPt: LPoint): TDialogImage;
001752      VAR s:                  TListScanner;
001753          dialogImage:        TDialogImage;
001754          currentDialogImage: TDialogImage;
001755  BEGIN
001756      {$IFC fTrace}BP(11);{$ENDC}
001757      currentDialogImage := NIL;
001758      IF SELF.Hit(mouseLPt) THEN
001759          BEGIN
001760          s := SELF.children.Scanner;
001761          WHILE s.Scan(dialogImage) DO
001762              IF dialogImage.isActive THEN
001763                  BEGIN
001764                  currentDialogImage := dialogImage.DownAt(mouseLPt);
001765                  IF currentDialogImage <> NIL THEN
001766                      s.Done;
001767                  END;
001768          END;
001769       DownAt := currentDialogImage;
001770      {$IFC fTrace}EP;{$ENDC}
001771  END;
```

```
001772
001773
001774  {$S DlgHot}
001775  PROCEDURE TDialog.PushButton(button: TButton);          {client or ToolKit may call}
001776      PROCEDURE TurnOnTheJuice;
001777          BEGIN
001778          button.Highlight(hOffToOn);
001779          END;
001780  BEGIN
001781      {$IFC fTrace}BP(11);{$ENDC}
001782      IF NOT button.isHighlighted THEN
001783          SELF.view.panel.OnAllPadsDo(TurnOnTheJuice);
001784      TDialogView(SELF.view).hitButton := button;
001785      SELF.ButtonPushed(button);
001786      {$IFC fTrace}EP;{$ENDC}
001787  END;
001788
001789
001790  {$S DlgWarm}
001791  PROCEDURE TDialog.RecalcExtent;
001792  BEGIN
001793      {$IFC fTrace}BP(11);{$ENDC}
001794      SUPERSELF.RecalcExtent;  {build up my size as the sum of the sizes of my children}
001795      IF SELF.parent = NIL THEN
001796          SELF.view.RecalcExtent;
001797      {$IFC fTrace}EP;{$ENDC}
001798  END;
001799
001800
001801  {$S DlgAlloc}
001802  PROCEDURE TDialog.SetDefaultButton(button: TButton);
001803      VAR thickPnSize:   point;
001804  BEGIN
001805      {$IFC fTrace}BP(11);{$ENDC}
001806      IF SELF.parent <> NIL THEN
001807          TDialog(SELF.parent).SetDefaultButton(button)
001808      ELSE
001809          TDialogView(SELF.view).SetDefaultButton(button);
001810      {$IFC fTrace}EP;{$ENDC}
001811  END;
001812
001813
001814  {$S DlgHot}
001815  PROCEDURE TDialog.SelectInputFrame(inputFrame: TInputFrame);
001816      VAR panel:          TPanel;
001817          newFrameSel:    TFrameSelection;
001818  BEGIN
001819      {$IFC fTrace}BP(11);{$ENDC}
```

```
001820      panel := SELF.view.panel;
001821      panel.BeginSelection;
001822      newFrameSel := TFrameSelection(panel.selection.FreedAndReplacedBy(
001823                           TFrameSelection.CREATE(NIL, SELF.Heap, inputFrame)));
001824      newFrameSel.coSelection.Become(
001825          inputFrame.textDialogImage.textImage.text.SelectAll(
001826                          inputFrame.textDialogImage.textImage));
001827      panel.Highlight(panel.selection.coSelection, hOffToOn);
001828      {$IFC fTrace}EP;{$ENDC}
001829      END;
001830
001831
001832  {$S DlgInit}
001833  END;
001834
001835
001836  {-----------------------------------------------------------------------------------------------}
001837
001838
001839  METHODS OF TButton;
001840
001841
001842  {$S DlgAlloc}
001843  FUNCTION TButton.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
001844           itsLocation: LPoint; itsMetrics: TButtonMetrics;
001845           sameSizedButton: TButton; itsCmdNumber: TCmdNumber):  TButton;
001846
001847      VAR buttonLRect:    LRect;
001848          itsLegend:      TLegend;
001849  BEGIN
001850      {$IFC fTrace}BP(11);{$ENDC}
001851      SetLRect(buttonLRect, -1, 0, 1, 1);
001852      OffsetLRect(buttonLRect, itsLocation.h, itsLocation.v);
001853      IF object = NIL THEN
001854          object := NewObject(heap, THISCLASS);
001855      SELF := TButton(TImageWithID.CREATE(object, heap, buttonLRect, itsId, itsView, TRUE));
001856
001857      WITH SELF DO
001858          BEGIN
001859          cmdNumber := itsCmdNumber;
001860          buttonMetrics := itsMetrics;
001861          isHighlighted := FALSE;
001862         {minWidth will be set by RecalcExtent}
001863          END;
001864
001865      IF sameSizedButton <> NIL THEN {weave me into chain of same-sized buttons}
001866          BEGIN
001867          SELF.nextSameSizedButton := sameSizedButton.nextSameSizedButton;
```

```
001868          sameSizedButton.nextSameSizedButton := SELF;
001869          END
001870      ELSE
001871          SELF.nextSameSizedButton := SELF;
001872
001873      itsLegend := NewStdLegend(heap, itsID, itsLocation.h, itsLocation.v, SELF.view,
001874                  itsMetrics.typeStyle);
001875      SELF.AddImage(itsLegend);
001876      SELF.Resize(buttonLRect);  {the AddImage will've made things out of balance}
001877      SELF.legend := itsLegend;
001878      itsLegend.wouldBeDraggable := FALSE; {as an entity unto itself during layout}
001879
001880      SELF.RecalcExtent;  {performs lots of magic}
001881
001882      {$IFC fTrace}EP;{$ENDC}
001883  END;
001884
001885
001886  {$IFC fDebugMethods}
001887  {$S DlgDbg}
001888  PROCEDURE TButton.Fields(PROCEDURE Field(nameAndType: S255));
001889  BEGIN
001890      SUPERSELF.Fields(Field);
001891      Field('cmdNumber: INTEGER');
001892      Field('minWidth: INTEGER');
001893      Field('isHighlighted: BOOLEAN');
001894      Field('nextSameSizedButton: TButton');
001895      Field('legend: TLegend');
001896      Field(CONCAT('buttonMetrics: RECORD height: INTEGER; curvH: INTEGER; curvV: INTEGER;',
001897              'typeStyle: LONGINT; expandNum: INTEGER; expandDen: INTEGER;',
001898              'absMinWidth: INTEGER; penState: ARRAY[1..18] OF Byte END'));
001899      Field('');
001900  END;
001901  {$ENDC}
001902
001903
001904  {$S DlgHot}
001905  PROCEDURE TButton.DrawJustMe;
001906  BEGIN
001907      {$IFC fTrace}BP(11);{$ENDC}
001908      SetPenState(SELF.buttonMetrics.penState);
001909      FrameLRRect(SELF.extentLRect, SELF.buttonMetrics.curvH, SELF.buttonMetrics.curvV);
001910      IF SELF.isHighlighted THEN
001911          InvrtLRRect(SELF.extentLRect, SELF.buttonMetrics.curvH, SELF.buttonMetrics.curvV);
001912      {$IFC fTrace}EP;{$ENDC}
001913  END;
001914
001915
```

```
001916  {$S DlgHot}
001917  PROCEDURE TButton.Highlight(highTransit: THighTransit);
001918  BEGIN
001919      {$IFC fTrace}BP(11);{$ENDC}
001920      InvrtLRRect(SELF.extentLRect, SELF.buttonMetrics.curvH, SELF.buttonMetrics.curvV);
001921      SELF.isHighLighted := (highTransit = hOffToOn);
001922      {$IFC fTrace}EP;{$ENDC}
001923  END;
001924
001925
001926  {$S DlgLayout}
001927  FUNCTION  TButton.LaunchLayoutBox(view: TView): TImage;
001928      VAR layoutBox:      TLayoutBox;
001929          layBoxExtent:   LRect;
001930          s:              TListScanner;
001931          childLayoutBox: TLayoutBox;
001932  BEGIN
001933      {$IFC fTrace}BP(11);{$ENDC}
001934      LaunchLayoutBox := TButtonLayoutBox.CREATE(NIL, SELF.Heap, SELF, view);
001935      {$IFC fTrace}EP;{$ENDC}
001936  END;
001937
001938
001939  {$S DlgHot}
001940  PROCEDURE TButton.MousePress(mouseLPt: LPoint);
001941      PROCEDURE TurnOnButton;
001942          BEGIN
001943          SELF.HighLight(hOffToOn);
001944          END;
001945  BEGIN
001946      {$IFC fTrace}BP(11);{$ENDC}
001947      SELF.view.panel.OnAllPadsDo(TurnOnButton);
001948      {$IFC fTrace}EP;{$ENDC}
001949  END;
001950
001951
001952  {$S DlgHot}
001953  PROCEDURE TButton.MouseRelease;
001954    BEGIN
001955        {$IFC fTrace}BP(11);{$ENDC}
001956        TDialog(SELF.parent).PushButton(SELF);
001957        {$IFC fTrace}EP;{$ENDC}
001958    END;
001959
001960
001961  {$S DlgAlloc}
001962  PROCEDURE TButton.RecalcExtent;
001963      VAR dialogView:         TDialogView;
```

```
001964          curWidth:           INTEGER;
001965          timesThrough:       INTEGER;
001966          nextButton:         TButton;
001967          legend:             TLegend;
001968          paraImage:          TParaImage;
001969          width:              INTEGER;
001970          styleIndex:         INTEGER;
001971          oldLegendLoc:       LPoint;
001972          lRectToInval:       LRect;
001973          legLength:          INTEGER;
001974          textDialogImage:    TTextDialogImage;
001975          editLegendSelection:TEditLegendSelection;
001976          paraExtent:         LRect;
001977  BEGIN
001978      {$IFC fTrace}BP(11);{$ENDC}
001979      lRectToInval := SELF.extentLRect;
001980      legend := TLegend(SELF.children.First);
001981      oldLegendLoc := legend.location;
001982
001983      WITH legend.extentLRect DO
001984          legLength := right - left;
001985      WITH SELF, buttonMetrics DO
001986  {$H-}   minWidth := MAX(absMinWidth, (legLength * expandNum) DIV expandDen);  {$H+}
001987
001988      curWidth := SELF.minWidth;
001989      FOR timesThrough := 1 TO 2 DO
001990          BEGIN
001991          nextButton := SELF.nextSameSizedButton;
001992          WHILE nextButton <> SELF DO {send this round my circle of same-sized buttons}
001993              BEGIN
001994              nextButton.Recompute(curWidth);
001995              WITH nextButton.extentLRect DO
001996                  curWidth := right - left;
001997              nextButton := nextButton.nextSameSizedButton;
001998              END;
001999          SELF.Recompute(curWidth);
002000          END;
002001
002002     UnionLRect(lRectToInval, SELF.extentLRect, lRectToInval);
002003     IF TDialogView(SELF.view).isShowing THEN
002004          SELF.view.panel.InvalLRect(lRectToInval);
002005
002006      IF SELF.parent <> NIL THEN
002007          SELF.parent.RecalcExtent;
002008      {$IFC fTrace}EP;{$ENDC}
002009  END;
002010
002011
```

```
002012  {$S DlgAlloc}
002013   PROCEDURE TButton.Recompute(minWidth: INTEGER);
002014      VAR buttonWidth:    INTEGER;
002015          labelWidth:     INTEGER;
002016          legend:         TLegend;
002017          topLeft:        LPoint;
002018          shape:          LRect;
002019          offset:         LPoint;
002020          width:          INTEGER;
002021          curLegendWidth: INTEGER;
002022          textExtent:     LRect;
002023          topCenter:      LPoint;
002024  BEGIN
002025      {$IFC fTrace}BP(11);{$ENDC}
002026      legend := SELF.legend;
002027
002028      topLeft := SELF.extentLRect.topLeft;
002029      WITH topCenter, SELF.extentLRect DO
002030          BEGIN
002031          v := top;
002032          h := (left + right) DIV 2;
002033          END;
002034
002035      buttonWidth := SELF.minWidth;
002036      IF buttonWidth < minWidth THEN
002037          buttonWidth := minWidth;
002038
002039      SetParaExtent(legend.paragraph, SELF.view, zeroLPt, textExtent);
002040      curLegendWidth := textExtent.right;
002041
002042      SetLPt(offset, (topCenter.h  - (curLegendWidth DIV 2)) - legend.location.h,
002043                     (topCenter.v +    ((SELF.buttonMetrics.height DIV 2) + 3)) - legend.location.v);
002044      legend.OffsetBy(offset);
002045
002046      SetLRect(shape, 0, 0, buttonWidth, SELF.buttonMetrics.height);
002047      OffSetLRect(shape, topCenter.h - (buttonWidth DIV 2), topCenter.v);
002048
002049      SELF.Resize(shape);
002050      {$IFC fTrace}EP;{$ENDC}
002051  END;
002052
002053
002054  {$S DlgHot}
002055  FUNCTION TButton.StillMyMouse(mouseLPt: LPoint): BOOLEAN;
002056  {Called when the mouse which went down in me has moved; possibly it is no longer in me}
002057      PROCEDURE TurnOffButton;
002058          BEGIN
002059          SELF.Highlight(hOnToOff);
```

```
002060          END;
002061  BEGIN
002062      {$IFC fTrace}BP(11);{$ENDC}
002063      IF SELF.Hit(mouseLPt) THEN
002064          StillMyMouse := TRUE {still in same button -- do nothing}
002065      ELSE { no longer in the button ; need to unhilight and remove claim }
002066          BEGIN
002067          SELF.view.panel.OnAllPadsDo(TurnOffButton);
002068          StillMyMouse := FALSE;  {see if anyone else wants this guy}
002069          END ;
002070      {$IFC fTrace}EP;{$ENDC}
002071  END;
002072
002073
002074  {$S DlgInit}
002075  END;
002076
002077
002078  {-------------------------------------------------------------------------------------------------}
002079
002080
002081  METHODS OF TCheckbox;
002082
002083
002084  {$S DlgAlloc}
002085  FUNCTION TCheckbox.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
002086              itsLocation: LPoint; boxWidth: INTEGER; boxHeight: INTEGER; wantLabel: BOOLEAN;
002087              labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox;
002088      VAR extentLRect:    LRect;
002089          tempLRect:      LRect;
002090          rectImage:      TRectImage;
002091          stringLoc:      LPoint;
002092          itsString:      TLegend;
002093  BEGIN
002094      {$IFC fTrace}BP(11);{$ENDC}
002095      SetLRect(extentLRect, 0, 0, boxWidth, boxHeight);
002096      OffsetLRect(extentLRect, itsLocation.h, itsLocation.v);
002097
002098      IF object = NIL THEN
002099          object := NewObject(heap, THISCLASS);
002100      SELF := TCheckbox(TImageWithID.CREATE(object, heap, extentLRect, itsId, itsView, TRUE));
002101
002102      SELF.isSelected := FALSE;
002103
002104      rectImage := TRectImage.CREATE(NIL, heap, extentLRect, noID, itsView, normalPen, FALSE);
002105      SELF.AddImage(rectImage);
002106      SELF.rectImage := rectImage;
002107
```

```
002108      IF wantLabel THEN
002109          BEGIN
002110          itsString := NewStdLegend(SELF.Heap, itsID, extentLRect.right + labelOffset.h,
002111                      extentLRect.bottom + labelOffset.v, itsView, itsTypeStyle);
002112          SELF.AddImage(itsString);
002113          SELF.legend := itsString;
002114          END
002115      ELSE
002116          SELF.legend := NIL;
002117      {$IFC fTrace}EP;{$ENDC}
002118  END;
002119
002120
002121  {$IFC fDebugMethods}
002122  {$S DlgDbg}
002123  PROCEDURE TCheckbox.Fields(PROCEDURE Field(nameAndType: S255));
002124  BEGIN
002125      SUPERSELF.Fields(Field);
002126      Field('isSelected: BOOLEAN');
002127      Field('rectImage: TRectImage');
002128      Field('legend: TLegend');
002129      Field('');
002130  END;
002131  {$ENDC}
002132
002133
002134  {$S DlgHot}
002135  PROCEDURE TCheckbox.ChangeLabel(newS255: S255);
002136  BEGIN
002137      {$IFC fTrace}BP(11);{$ENDC}
002138      IF SELF.legend = NIL THEN
002139  {$IFC fDbgOK}
002140          ABCBreak('No legend to chg', 0)  {later could perhaps launch a new label}
002141  {$ENDC}
002142      ELSE
002143          SELF.legend.ChangeString(newS255);
002144      SELF.RecalcExtent;
002145      {$IFC fTrace}EP;{$ENDC}
002146  END;
002147
002148
002149  {$S DlgHot}
002150  FUNCTION TCheckbox.CursorAt(mouseLPt: LPoint): TCursorNumber;
002151  BEGIN
002152      {$IFC fTrace}BP(11);{$ENDC}
002153      IF SELF.Hit(mouseLPt) THEN
002154          CursorAt := checkCursor
002155      ELSE
```

```
002156          CursorAt := noCursor;
002157      {$IFC fTrace}EP;{$ENDC}
002158  END;
002159
002160
002161  {$S DlgHot}
002162  PROCEDURE TCheckbox.Draw;
002163  BEGIN
002164      {$IFC fTrace}BP(11);{$ENDC}
002165      PenNormal;
002166      IF SELF.IsSelected THEN
002167          FillLRect(SELF.rectImage.extentLRect, lPatBlack)     {fill with black if selected}
002168      ELSE
002169          FillLRect(SELF.rectImage.extentLRect, lPatWhite);
002170      SELF.rectImage.Draw;                      {draw the outline box in any case}
002171      IF SELF.legend <> NIL THEN
002172          SELF.legend.Draw;
002173      {$IFC fTrace}EP;{$ENDC}
002174  END;
002175
002176
002177  {$S DlgLayout}
002178  FUNCTION  TCheckbox.LaunchLayoutBox(view: TView): TImage;
002179      VAR layoutBox:      TLayoutBox;
002180          childLayoutBox: TLayoutBox;
002181  BEGIN
002182      {$IFC fTrace}BP(11);{$ENDC}
002183      IF SELF.legend <> NIL THEN {has string; use normal layout box}
002184          LaunchLayoutBox := SUPERSELF.LaunchLayoutBox(view)
002185      ELSE
002186          BEGIN  {a checkbox without an associated string}
002187          childLayoutBox := TLayoutBox(SELF.rectImage.LaunchLayoutBox(view));
002188          layoutBox := TLayoutBox.CREATE(NIL, SELF.Heap, childLayoutBox.extentLRect, noID, NIL,
002189              view, SELF, zeroRect, FALSE, FALSE, TRUE);
002190          layoutBox.AddImage(childLayoutBox);
002191          LaunchLayoutBox := layoutBox;
002192          END;
002193      {$IFC fTrace}EP;{$ENDC}
002194  END;
002195
002196
002197  {$S DlgHot}
002198  PROCEDURE TCheckbox.MousePress(mouseLPt: LPoint);
002199  {This proc is only called for mouspresses within "free check boxes", which is to say Checkboxes
002200  which are componenents of a dialogView -- NOT for Checkboxes which are subdialogImages of another dialogImage}
002201      VAR carryOutTheToggle:  BOOLEAN;
002202          dialogView:         TDialogView;
002203  BEGIN
```

```
002204      {$IFC fTrace}BP(11);{$ENDC}
002205      carryOutTheToggle := TRUE;
002206      dialogView := TDialogView(SELF.view);
002207      IF dialogView.paintFreeBoxes THEN {need to worry about only toggling if in current sense}
002208          BEGIN
002209          IF dialogView.startedPainting THEN {have already started 'painting' free checkboxes}
002210              BEGIN
002211              IF dialogView.paintSense = SELF.isSelected THEN
002212                  carryOutTheToggle := FALSE {already in the sense being painted--do nothing}
002213              END
002214          ELSE {just starting painting; establish the painting sense}
002215              BEGIN
002216              dialogView.startedPainting := TRUE;
002217              dialogView.paintSense := NOT SELF.isSelected;
002218              END;
002219          END;
002220      IF carryOutTheToggle THEN
002221         BEGIN
002222         SELF.Toggle;
002223         dialogView.panel.InvalLRect(SELF.rectImage.extentLRect);
002224         SELF.ControlHit(SELF, SELF.isSelected); {pass it up to cluster and Dialog and even DialogView}
002225         END;
002226      {$IFC fTrace}EP;{$ENDC}
002227  END;
002228
002229
002230  {$S DlgHot}
002231  PROCEDURE TCheckbox.Toggle;
002232  BEGIN
002233      {$IFC fTrace}BP(11);{$ENDC}
002234      SELF.isSelected := NOT SELF.isSelected;
002235      {$IFC fTrace}EP;{$ENDC}
002236  END;
002237
002238
002239  {$S DlgInit}
002240  END;
002241
002242
002243  {-----------------------------------------------------------------------------------------------}
002244
002245  METHODS OF TCluster;
002246
002247
002248  {$S DlgAlloc}
002249  FUNCTION  TCluster.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
002250              itsLocation: LPoint): TCluster;
002251                  VAR extentLRect: LRect;
```

```
002252  BEGIN
002253      {$IFC fTrace}BP(11);{$ENDC}
002254      WITH itsLocation DO
002255          SetLRect(extentLRect, h, v, h + 1, v + 1); {include that pt in ultimate extent}
002256      IF object = NIL THEN
002257          object := NewObject(heap, THISCLASS);
002258      SELF := TCluster(TImageWithID.CREATE(object, heap, extentLRect, itsId, itsView, TRUE));
002259
002260      WITH SELF DO
002261          BEGIN
002262          location := itsLocation;
002263          hitBox := NIL;
002264          hiLitBox := NIL;
002265          lastBox  := NIL;
002266          END;
002267      {$IFC fTrace}EP;{$ENDC}
002268  END;
002269
002270
002271  {$IFC fDebugMethods}
002272  {$S DlgDbg}
002273  PROCEDURE TCluster.Fields(PROCEDURE Field(nameAndType: S255));
002274  BEGIN
002275      SUPERSELF.Fields(Field);
002276      Field('location: LPoint');
002277      Field('hitBox: TCheckBox');
002278      Field('hiLitBox: TCheckBox');
002279      Field('lastBox: TCheckBox');
002280      Field('');
002281  END;
002282  {$ENDC}
002283
002284
002285  {$S DlgAlloc}
002286  FUNCTION TCluster.AddAlignedCheckbox(itsId: S255; selectThisOne: BOOLEAN): TCheckbox;
002287      CONST   stdIncrement = 20;
002288      VAR lastBox:        TCheckbox;
002289          location:       LPoint;
002290          itsBoxWidth:    INTEGER;
002291          itsBoxHeight:   INTEGER;
002292          checkBox:       TCheckbox;
002293          wantLabel:      BOOLEAN;
002294          labelOffset:    Point;
002295          vhs:            VHSelect;
002296          itsTypeStyle:   TTypeStyle;
002297          styleChange:    TStyleChange;
002298  BEGIN
002299      {$IFC fTrace}BP(11);{$ENDC}
```

```
002300      itsTypeStyle := sysTypeStyle;
002301      wantLabel := (itsID <> noID);
002302      labelOffset := stdLabelOffset;
002303      itsBoxWidth := stdBoxWidth;
002304      itsBoxHeight := stdBoxHeight;
002305
002306      lastBox := SELF.lastBox;
002307      IF lastBox = NIL THEN {this is the first to be inserted}
002308          location := SELF.location
002309      ELSE    {there is already at least one box in the cluster -- align to right of it & use its params}
002310          BEGIN
002311          WITH location DO
002312              BEGIN
002313              h := lastBox.extentLRect.right + stdIncrement;
002314              v := lastBox.rectImage.extentLRect.top;
002315              END;
002316          WITH lastBox.rectImage.extentLRect DO
002317              BEGIN
002318              itsBoxWidth := right - left;
002319              itsBoxHeight := bottom - top;
002320              END;
002321          IF wantLabel THEN
002322              BEGIN
002323              IF lastBox.legend <> NIL THEN   {use same type style and label Offset as prev guy}
002324                  BEGIN
002325                  lastBox.legend.paragraph.typeStyles.GetAt(1, @styleChange);
002326                  WITH lastBox, legend DO
002327                      BEGIN
002328                      itsTypeStyle := styleChange.newStyle;
002329                      FOR vhs := v TO h DO
002330                          labelOffset.vh[vhs] := location.vh[vhs] -
002331                              rectImage.extentLRect.botRight.vh[vhs];
002332                      END;
002333                  END
002334              ELSE
002335                  wantLabel := FALSE;  {last box did not have a label, so I won't either}
002336              END;
002337          END;
002338
002339      checkBox := SELF.AddCheckbox(itsID, location, itsBoxWidth, itsBoxHeight,
002340                  wantLabel, labelOffset, itsTypeStyle, selectThisOne);
002341      IF lastBox <> NIL THEN
002342          checkbox.idNumber := lastBox.idNumber + 1;
002343      AddAlignedCheckbox := checkbox;
002344
002345      {$IFC fTrace}EP;{$ENDC}
002346  END;
002347
```

```
002348
002349  {$S DlgAlloc}
002350  FUNCTION TCluster.NewAlignedCheckbox(itsPhrase: INTEGER; selectThisOne: BOOLEAN): TCheckbox;
002351      VAR itsID:          S255;
002352          itsLocation:    LPoint;
002353          checkbox:       TCheckbox;
002354  BEGIN
002355      {$IFC fTrace}BP(11);{$ENDC}
002356      GetTextAndLocation(itsPhrase, itsID, itsLocation);
002357      checkbox := SELF.AddAlignedCheckbox(itsID, selectThisOne);
002358      checkbox.idNumber := itsPhrase;
002359      NewAlignedCheckbox := checkbox;
002360      {$IFC fTrace}EP;{$ENDC}
002361  END;
002362
002363
002364  {$S DlgAlloc}
002365  FUNCTION TCluster.AddCheckbox(itsID: S255; itsLocation: LPoint; boxWidth: INTEGER;
002366                        boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point;
002367                        itsTypeStyle: TTypeStyle;  selectThisOne: BOOLEAN): TCheckbox;
002368      VAR location:   LPoint;
002369          checkbox:   TCheckbox;
002370  BEGIN
002371      {$IFC fTrace}BP(11);{$ENDC}
002372      checkbox := TCheckbox.CREATE(NIL, SELF.Heap, itsID, SELF.view, itsLocation, boxWidth,
002373                      boxHeight, wantLabel, labelOffset, itsTypeStyle);
002374      SELF.AddImage(checkbox);
002375      SELF.lastBox := checkbox;
002376      IF selectThisOne THEN
002377          BEGIN
002378          IF NOT checkbox.isSelected THEN
002379              checkbox.Toggle;
002380          SELF.hiLitBox := checkbox;
002381          END;
002382      AddCheckbox := checkbox;
002383      {$IFC fTrace}EP;{$ENDC}
002384  END;
002385
002386
002387  {$S DlgAlloc}
002388  FUNCTION TCluster.NewCheckbox(itsPhrase: INTEGER; boxWidth: INTEGER;
002389                      boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle;
002390                      selectThisOne: BOOLEAN): TCheckbox;
002391      VAR itsID:          S255;
002392          itsLocation:    LPoint;
002393          checkbox:       TCheckbox;
002394  BEGIN
002395      {$IFC fTrace}BP(11);{$ENDC}
```

```
002396      GetTextAndLocation(itsPhrase, itsID, itsLocation);
002397      checkbox := SELF.AddCheckbox(itsID, itsLocation, boxWidth, boxHeight, wantLabel,  labelOffset,
002398          itsTypeStyle, selectThisOne);
002399      checkbox.idNumber := itsPhrase;
002400      NewCheckbox := checkbox;
002401      {$IFC fTrace}EP;{$ENDC}
002402  END;
002403
002404
002405  {$S DlgAlloc}
002406  PROCEDURE TCluster.AddRowOfBoxes(numberOfBoxes: INTEGER; startingIDNumber: INTEGER;
002407                  boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER);
002408      VAR currentIDNumber: INTEGER;
002409          checkbox:       TCheckbox;
002410          newLocation:    LPoint;
002411          newID:          S255;
002412  BEGIN
002413      {$IFC fTrace}BP(11);{$ENDC}
002414      FOR currentIDNumber := startingIDNumber TO (startingIDNumber + numberOfBoxes -  1) DO
002415          BEGIN
002416          IF SELF.lastBox = NIL THEN {this is the first to be inserted}
002417              newLocation := SELF.location
002418          ELSE { There is already at least one box in the cluster -- align to right of it}
002419              WITH newLocation DO
002420                  BEGIN
002421                  h := SELF.lastBox.rectImage.extentLRect.right + boxSpacing;  {??}
002422                  v := SELF.lastBox.rectImage.extentLRect.top;
002423                  END;
002424          IntToString(currentIDNumber, @newID);
002425          checkbox := SELF.AddCheckbox(newID, newLocation, boxWidth, boxHeight, FALSE, zeroPt,
002426              sysTypeStyle, FALSE);
002427          checkBox.IDNumber := currentIDNumber;
002428          END;
002429      {$IFC fTrace}EP;{$ENDC}
002430  END;
002431
002432
002433  {$S DlgHot}
002434  FUNCTION TCluster.Hit(mouseLPt: LPoint): BOOLEAN;
002435      VAR checkbox:   TCheckbox;
002436          s:          TListScanner;
002437  BEGIN
002438      {$IFC fTrace}BP(11);{$ENDC}
002439      Hit := FALSE;
002440      IF LRectHasLPt(SELF.extentLRect, mouseLPt) THEN  {passes coarsest hit test; look more deeply now}
002441          BEGIN
002442          s := SELF.children.Scanner;
002443          WHILE s.Scan(checkbox) DO
```

```
002444              IF checkbox.Hit(mouseLPt) THEN
002445                  BEGIN
002446                  Hit := TRUE;
002447                  SELF.hitBox := checkbox;
002448                  s.Done;
002449                  END;
002450          END;
002451      {$IFC fTrace}EP;{$ENDC}
002452  END;
002453
002454
002455  {$S DlgHot}
002456  PROCEDURE TCluster.MousePress(mouseLPt: LPoint);
002457      {We are assured that when this is called, it will have been immediately
002458        preceded by a successful call to TCluster.Hit .  Hence, the field
002459        TCluster.hitBox will correctly point to which guy was hit.}
002460  BEGIN
002461      {$IFC fTrace}BP(11);{$ENDC}
002462      SELF.SelectBox(SELF.hitBox);  {will deselect any other}
002463      {$IFC fTrace}EP;{$ENDC}
002464  END;
002465
002466
002467  {$S DlgHot}
002468  PROCEDURE TCluster.SelectBox(checkbox: TCheckbox); {select this box, deselecting others}
002469          PROCEDURE DrawUnHiLitBoxOnThePad;
002470          BEGIN
002471              SELF.hiLitBox.Draw; {redraw old box unhilit}
002472          END;
002473
002474          PROCEDURE DrawHiLitBoxOnThePad;
002475          BEGIN
002476              checkbox.Draw;     {toggle the newly selected one on}
002477          END;
002478  BEGIN
002479      {$IFC fTrace}BP(11);{$ENDC}
002480      IF (SELF.hiLitBox <> checkbox) AND (checkbox <> NIL) THEN
002481          BEGIN
002482          IF SELF.hiLitBox <> NIL THEN
002483              BEGIN
002484              SELF.hiLitBox.Toggle;
002485              SELF.view.panel.OnAllPadsDo(DrawUnHiLitBoxOnThePad);
002486              SELF.ControlHit(SELF.hiLitBox, FALSE);
002487              END;
002488          SELF.hiLitBox := checkbox;   {set new box as the currently hilit one}
002489          checkbox.Toggle;
002490          SELF.view.panel.OnAllPadsDo(DrawHiLitBoxOnThePad);
002491          SELF.ControlHit(checkbox, TRUE);
```

```
002492          END;
002493      {$IFC fTrace}EP;{$ENDC}
002494  END;
002495
002496
002497  {$S DlgHot}
002498  FUNCTION  TCluster.StillMyMouse(mouseLPt: LPoint): BOOLEAN;
002499  BEGIN
002500      {$IFC fTrace}BP(11);{$ENDC}
002501      IF SELF.Hit(mouseLPt) THEN {mouse is in a box of the cluster}
002502          BEGIN
002503          SELF.SelectBox(SELF.hitBox);   {will toggle any alternate box off}
002504          StillMyMouse := TRUE;
002505          END
002506      ELSE {mouse not in any of my box's hit areas at the moment}
002507          StillMyMouse := FALSE;
002508      {$IFC fTrace}EP;{$ENDC}
002509  END;
002510
002511
002512  {$S DlgInit}
002513  END;
002514
002515
002516  METHODS OF TInputFrame;
002517
002518
002519  {$S DlgAlloc}
002520  FUNCTION TInputFrame.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
002521          promptLocation: LPoint; promptTypeStyle: TTypeStyle;
002522          inputLocation:  LPoint; inputTypeStyle: TTypeStyle;  maxInputChars: INTEGER;
002523          itsBorders: Rect; drawInputLRect: BOOLEAN; drawHitLRect: BOOLEAN): TInputFrame;
002524      VAR textExtent:         LRect;
002525          myOwnExtentLRect:   LRect;
002526          prompt:             TLegend;
002527          textDialogImage:    TTextDialogImage;
002528  {$IFC libraryVersion <= 20}   {* * * P E P S I * * *}
002529          fInfo:              TFInfo;
002530  {$ELSEC}                      {* *  S P R I N G  * *}
002531          fInfo:              FontInfo;
002532  {$ENDC}
002533  BEGIN
002534      {$IFC fTrace}BP(11);{$ENDC}
002535      prompt := TLegend.CREATE(NIL, heap, itsID, itsView, promptLocation, promptTypeStyle);
002536
002537      SetQDTypeStyle(inputTypeStyle);
002538      GetFontInfo(fInfo);
002539      WITH fInfo DO
```

```
002540          SetLRect(textExtent, 0, -ascent - leading, maxInputChars * widMax,  descent + leading);
002541
002542      OffsetLRect(textExtent, inputLocation.h, inputLocation.v);
002543      textDialogImage := TTextDialogImage.CREATE(NIL, heap, textExtent, 'input', itsView,
002544          inputTypeStyle, '');
002545
002546      UnionLRect(prompt.extentLRect, textDialogImage.extentLRect, myOwnExtentLRect);
002547      LRectAddBorders(myOwnExtentLRect, itsBorders, myOwnExtentLRect);
002548
002549      IF object = NIL THEN
002550          object := NewObject(heap, THISCLASS);
002551      SELF := TInputFrame(TImageWithID.CREATE(object, heap, myOwnExtentLRect, itsId, itsView, TRUE));
002552
002553      SELF.prompt := prompt;
002554      SELF.AddImage(prompt);
002555
002556      SELF.textDialogImage := textDialogImage;
002557      SELF.AddImage(textDialogImage);
002558
002559      SELF.inputTypeStyle := inputTypeStyle;
002560      SELF.maxInputChars := maxInputChars;
002561      SELF.drawHitLRect := drawHitLRect;
002562      SELF.drawInputLRect := drawInputLRect;
002563      SELF.borders := itsBorders;
002564
002565      {$IFC fTrace}EP;{$ENDC}
002566  END;
002567
002568
002569  {$IFC fDebugMethods}
002570  {$S DlgDbg}
002571  PROCEDURE TInputFrame.Fields(PROCEDURE Field(nameAndType: S255));
002572  BEGIN
002573      SUPERSELF.Fields(Field);
002574      Field('textDialogImage: TTextDialogImage');
002575      Field('prompt: TLegend');
002576      Field('borders: Rect');
002577      Field('drawInputLRect: BOOLEAN');
002578      Field('drawHitLRect: BOOLEAN');
002579      Field('maxInputChars: INTEGER');
002580      Field('inputTypeStyle: LONGINT');   {make this right someday}
002581      Field('');
002582  END;
002583  {$ENDC}
002584
002585
002586  {$S DlgText}
002587  FUNCTION TInputFrame.CursorAt(mouseLPt: LPoint): TCursorNumber;
```

```
002588      BEGIN
002589          {$IFC fTrace}BP(11);{$ENDC}
002590          IF SELF.Hit(mouseLPt) THEN
002591              CursorAt := textCursor
002592          ELSE
002593              CursorAt := noCursor;
002594          {$IFC fTrace}EP;{$ENDC}
002595      END;
002596
002597
002598  {$S DlgText}
002599  PROCEDURE TInputFrame.Draw;
002600          VAR tempLRect:  LRect;
002601              pat:        pattern;
002602      BEGIN
002603          {$IFC fTrace}BP(11);{$ENDC}
002604          IF SELF.prompt <> NIL THEN
002605              SELF.prompt.Draw;
002606          SELF.textDialogImage.Draw;    {draw the current input characters lying there...}
002607          IF SELF.drawInputLRect THEN
002608              BEGIN
002609              tempLRect := SELF.textDialogImage.textImage.extentLRect;
002610              InsetLRect(tempLRect, -6, -4);
002611              PenNormal;
002612              thePad.LPatToPat(lPatGray, pat);
002613              PenPat(pat);
002614              PenSize(1,1);
002615              FrameLRect(tempLRect);  {mostly for debugging reassurance...}
002616              END;
002617          IF SELF.drawHitLRect THEN
002618              FrameLRect(SELF.extentLRect);
002619          {$IFC fTrace}EP;{$ENDC}
002620      END;
002621
002622
002623  {$S DlgLayout}
002624  FUNCTION  TInputFrame.LaunchLayoutBox(view: TView): TImage;
002625  {In the future, if there were one, we would want to allow resizing of the hit area during
002626    layout, and would here launch a special type of layout box, TInptFrmLayoutBox, to do layout just right}
002627      VAR layoutBox:      TLayoutBox;
002628          layBoxExtent:   LRect;
002629          s:              TListScanner;
002630          childLayoutBox: TLayoutBox;
002631          newBorders:     Rect;
002632  BEGIN
002633      {$IFC fTrace}BP(11);{$ENDC}
002634      layoutBox := TLayoutBox(SUPERSELF.LaunchLayoutBox(view));  {i.e., TImageWithID's launch}
002635      WITH layoutBox.borders DO
```

```
002636          BEGIN
002637          top := SELF.borders.top;
002638          right := SELF.borders.right - right;
002639          bottom := SELF.borders.bottom - bottom;
002640          left := SELF.borders.left - left;
002641          END;
002642      layoutBox.RecalcExtent;
002643      LaunchLayoutBox := layoutBox;
002644
002645      {$IFC fTrace}EP;{$ENDC}
002646  END;
002647
002648
002649  {$S DlgText}
002650  PROCEDURE TInputFrame.MousePress(mouseLPt: LPoint);
002651      VAR frameSelection: TFrameSelection;
002652  BEGIN
002653      {$IFC fTrace}BP(11);{$ENDC}
002654       LRectHaveLPt(SELF.textDialogImage.textImage.extentLRect, mouseLPt);
002655       TDialogView(SELF.view).magnetCursor := textCursor;
002656       frameSelection := TFrameSelection(SELF.view.panel.selection.FreedAndReplacedBy(
002657                                           TFrameSelection.CREATE(NIL, SELF.Heap, SELF)));
002658       SELF.textDialogImage.textImage.MousePress(mouseLPt);
002659      {$IFC fTrace}EP;{$ENDC}
002660  END;
002661
002662
002663  {$S DlgText}
002664  PROCEDURE TInputFrame.GetContents(VAR theStr: S255);
002665      VAR paraImage:  TParaImage;
002666          paragraph:  TParagraph;
002667  BEGIN
002668      {$IFC fTrace}BP(11);{$ENDC}
002669      paraImage := TParaImage(SELF.textDialogImage.textImage.imageList.First);
002670      paragraph := paraImage.paragraph;
002671      paragraph.ToPStr(@theStr);
002672      {$IFC fTrace}EP;{$ENDC}
002673  END;
002674
002675
002676  {$S DlgLayout}
002677  PROCEDURE TInputFrame.RecalcExtent;
002678      VAR newExtent:  LRect;
002679      BEGIN
002680          {$IFC fTrace}BP(11);{$ENDC}
002681          IF SELF.prompt <> NIL THEN
002682              UnionLRect(SELF.prompt.extentLRect, SELF.textDialogImage.extentLRect, newExtent)
002683          ELSE
```

```
002684              newExtent := SELF.textDialogImage.extentLRect;
002685          LRectAddBorders(newExtent, SELF.borders, newExtent);
002686
002687          SELF.Resize(newExtent);
002688          IF SELF.parent <> NIL THEN
002689              SELF.parent.RecalcExtent;
002690
002691          {$IFC fTrace}EP;{$ENDC}
002692      END;
002693
002694
002695  {$S DlgText}
002696  FUNCTION TInputFrame.StillMyMouse(mouseLPt: LPoint): BOOLEAN;
002697    {in this implementation, once the insertion point has been dropped, we don't give up
002698      control even if user now strays outside our hit area}
002699      BEGIN
002700          {$IFC fTrace}BP(11);{$ENDC}
002701           LRectHaveLPt(SELF.textDialogImage.textImage.extentLRect, mouseLPt);
002702           SELF.view.panel.selection.coSelection.MouseMove(mouseLPt);  {currently, just pass it on to the
002703                                                                        text selection}
002704           StillMyMouse := TRUE;
002705          {$IFC fTrace}EP;{$ENDC}
002706      END;
002707
002708
002709  {$S DlgText}
002710  PROCEDURE TInputFrame.SupplantContents(newStr: S255);
002711      VAR paragraph:  TParagraph;
002712          paraImage:  TParaImage;
002713          textImage:  TTextImage;
002714          oldCount:   INTEGER;
002715      BEGIN
002716          {$IFC fTrace}BP(11);{$ENDC}
002717          textImage := SELF.textDialogImage.textImage;
002718          paraImage := TParaImage(textImage.imageList.First);
002719          paragraph := paraImage.paragraph;
002720          oldCount := paragraph.size;
002721          paragraph.ReplPString(0, oldCount, @newStr);
002722
002723          paraImage.changed := TRUE;
002724          paraImage.InvalLinesWith(0, MAXINT);
002725
002726          textImage.RecomputeImages(actionNone, TRUE);
002727          IF TDialogView(SELF.view).isShowing THEN
002728              SELF.view.panel.InvalLRect(SELF.textDialogImage.extentLRect);
002729          {$IFC fTrace}EP;{$ENDC}
002730      END;
002731
```

```
002732
002733  {$S DlgInit}
002734  END;
002735
002736
002737  METHODS OF TLegend;
002738
002739  {$S TK2Start}
002740  FUNCTION  TLegend.CREATE(object: TObject; heap: THeap; itsChars: S255; itsView: TView;
002741              itsLocation: LPoint; itsTypeStyle: TTypeStyle):  TLegend;
002742      VAR itsExtent:      LRect;
002743          height:         INTEGER;
002744          itsParagraph:   TParagraph;
002745  BEGIN
002746      {$IFC fTrace}BP(11);{$ENDC}
002747      itsParagraph := TParagraph.CREATE(NIL, heap, LENGTH(itsChars), itsTypeStyle);
002748      itsParagraph.InsPStrAt(1, @itsChars);
002749      SetParaExtent(itsParagraph, itsView, itsLocation, itsExtent);
002750
002751      IF object = NIL THEN
002752          object := NewObject(heap, THISCLASS);
002753      SELF := TLegend(TDialogImage.CREATE(object, heap, itsExtent, noID, itsView, FALSE));
002754
002755      WITH SELF DO
002756          BEGIN
002757          location := itsLocation;
002758          paragraph := itsParagraph;
002759          wouldBeDraggable := TRUE;
002760          usesSysFont := (itsTypeStyle.font.fontFamily = famSystem);
002761          END;
002762
002763      {$IFC fTrace}EP;{$ENDC}
002764  END;
002765
002766
002767  {$S DlgCold}
002768  PROCEDURE TLegend.Free;
002769  BEGIN
002770      {$IFC fTrace}BP(11);{$ENDC}
002771      Free(SELF.paragraph);
002772      SUPERSELF.Free;
002773      {$IFC fTrace}EP;{$ENDC}
002774  END;
002775
002776
002777  {$IFC fDebugMethods}
002778  {$S DlgDbg}
002779  PROCEDURE TLegend.Fields(PROCEDURE Field(nameAndType: S255));
```

```
002780  BEGIN
002781      SUPERSELF.Fields(Field);
002782      Field('location: LPoint');
002783      Field('paragraph: TParagraph');
002784      Field('wouldBeDraggable: BOOLEAN');
002785      Field('usesSysFont: BOOLEAN');
002786      Field('');
002787  END;
002788  {$ENDC}
002789
002790
002791  {$S DlgHot}
002792  PROCEDURE  TLegend.ChangeToPhrase(newPhrase: INTEGER);
002793      VAR newString:  S255;
002794  BEGIN
002795      {$IFC fTrace}BP(11);{$ENDC}
002796      process.GetAlert(newPhrase, newString);
002797      SELF.ChangeString(newString);
002798      {$IFC fTrace}EP;{$ENDC}
002799  END;
002800
002801
002802  {$S DlgHot}
002803  PROCEDURE  TLegend.ChangeString(newString: S255);
002804  BEGIN
002805      {$IFC fTrace}BP(11);{$ENDC}
002806      SELF.view.panel.InvalLRect(SELF.extentLRect); {invalidate old string's bounding box}
002807      SELF.paragraph.DelAll;
002808      SELF.paragraph.InsPStrAt(1, @newString);
002809      SELF.GetBoxRight;
002810      SELF.view.panel.InvalLRect(SELF.extentLRect); {invalidate new string's bounding box}
002811      {$IFC fTrace}EP;{$ENDC}
002812  END;
002813
002814
002815  {$S DlgHot}
002816  PROCEDURE TLegend.Draw;
002817  BEGIN
002818      {$IFC fTrace}BP(11);{$ENDC}
002819      MoveToL(SELF.location.h, SELF.location.v);
002820      SELF.paragraph.Draw(1, SELF.paragraph.size);
002821      {$IFC fTrace}EP;{$ENDC}
002822  END;
002823
002824
002825  {$S TK2Start}
002826  PROCEDURE TLegend.GetBoxRight;
002827      VAR newExtent:      LRect;
```

```
002828  BEGIN
002829      {$IFC fTrace}BP(11);{$ENDC}
002830      SetParaExtent(SELF.paragraph, SELF.view, SELF.location, newExtent);
002831      SELF.Resize(newExtent);
002832      {$IFC fTrace}EP;{$ENDC}
002833  END;
002834
002835
002836  {$S DlgHot}
002837  PROCEDURE TLegend.GetString(VAR itsString: S255);
002838  BEGIN
002839      {$IFC fTrace}BP(11);{$ENDC}
002840      SELF.paragraph.ToPStr(@itsString);
002841      {$IFC fTrace}EP;{$ENDC}
002842  END;
002843
002844
002845  {$S SgLayout}
002846  FUNCTION  TLegend.LaunchLayoutBox(view: TView): TImage;
002847  BEGIN
002848      {$IFC fTrace}BP(11);{$ENDC}
002849      IF SELF.isEditable THEN
002850          LaunchLayoutBox := TLegendLayoutBox.CREATE(NIL, SELF.Heap, view, SELF)
002851      ELSE
002852          LaunchLayoutBox := SUPERSELF.LaunchLayoutBox(view);
002853      {$IFC fTrace}EP;{$ENDC}
002854  END;
002855
002856
002857  {$S DlgHot}
002858  PROCEDURE TLegend.OffsetBy(deltaLPt: LPoint);
002859  BEGIN
002860      {$IFC fTrace}BP(11);{$ENDC}
002861  {$H-} LPtPlusLPt(SELF.location, deltaLPt, SELF.location); {$H+}
002862      SUPERSELF.OffsetBy(deltaLPt);
002863      {$IFC fTrace}EP;{$ENDC}
002864  END;
002865
002866
002867  {$S DlgHot}
002868  PROCEDURE TLegend.RecalcExtent;
002869  BEGIN
002870      {$IFC fTrace}BP(11);{$ENDC}
002871      SELF.GetBoxRight;
002872      IF SELF.parent <> NIL THEN
002873          SELF.parent.RecalcExtent;
002874      {$IFC fTrace}EP;{$ENDC}
002875  END;
```

```
002876
002877
002878  {$S DlgInit}
002879  END;

End of File -- Lines: 2879 Characters: 81055
```

```
================================================================================================
FILE: "LIBTK/UDIALOG3.TEXT"
================================================================================================

000001  {UDialog3}  {COPYRIGHT 1984 BY APPLE COMPUTER, INC}
000002
000003  (*
000004      TPicObject  -- TRectImage -- TTextDialogImage - TFrameSelection  -
000005      TPlannerView - TLayoutBox - TLegendLayoutBox - ButtonLayoutBox - TTitleTab -
000006      TLayPickSelection - TLayMoveCommand - TEditLegendSelection  - TDialogDesignWindow
000007
000008  *)
000009
000010  {04/25/84 1610  Switched back to using a paraImage in call to FilterAndDo, as per JKD's latest change,
000011                     in TLegendLayoutBox.RecalcExtent
000012                  Removed the inval in TLegendLayoutBox.MousePress, fExperimenting or not}
000013  {04/25/84 0015  Added TEditLegendSelection.MousePress, MouseMove and MouseRelease and field tripleClick,
000014                      to trap triple-click and do a SelectAll with it}
000015  {04/23/84 1210  Removed all references to 'underEdit' field of TDialogImage
000016                  TEditLegendSelection.Deselect, Free, and Restore changed.
000017                  Removed some commented-out code and some unused VAR declarations in the TEditLegendSel
000018                     methods changed}
000019  {04/17/84 2130  In TEditLegendSelection.CREATE doesn't inval unless fExperimenting
000020                  Removed ABCBreak calls in TEditLegendSelection.CREATE, TFrameSelection.KeyChar
000021                  TLegendLayoutBox.Draw always keys on existence of SELF.textDialogImage, ignoring
000022                       underEdit flag; underEdit, if this is okay, can vanish completely from the architecture}
000023  {04/17/84 2000  In TLegendLayoutBox.RecalcExtent, try to use TImage.FilterAndDo correctly}
000024  {04/15/84 2000  Spring Prelim Release}
000025  {01/29/84 1754  RELEASE TK8D}
000026  {12/22/83 1927  RELEASE TK8A}
000027
000028
000029  METHODS OF TPicObject;
000030
000031  {$S DlgCold}
000032  FUNCTION  TPicObject.CREATE(object: TObject; heap: THeap; itsId: S255;
000033              itsView: TView; itsLocation: LPoint; itsPicHandle: PicHandle):  TPicObject;
000034  VAR tempHz:         THeap;
000035      frameInView:    LRect;
000036      myPicHandle:    PicHandle;
000037      boxAtCreation:  Rect;
000038  BEGIN
000039      {$IFC fTrace}BP(7);{$ENDC}
000040      boxAtCreation:= itsPicHandle^^.picFrame;
000041      noPad.rectToLRect(boxAtCreation, frameInView);
000042      OffsetLRect(frameInView, itsLocation.h, itsLocation.v);
000043
```

```
000044      IF object = NIL THEN
000045         object := NewObject(heap, THISCLASS);
000046      SELF := TPicObject(TImageWithID.CREATE(object, heap, frameInView, itsId, itsView, FALSE {no children}));
000047
000048      SELF.isEditable := FALSE;
000049      SELF.boxAtCreation := boxAtCreation;
000050      GetHeap(tempHz);
000051      SetHeap(SELF.Heap);
000052      myPicHandle := OpenPicture(SELF.boxAtCreation);   {replay the incoming picture file onto our own heap}
000053      SELF.picture := myPicHandle;
000054      DrawPicture(itsPicHandle, SELF.boxAtCreation);
000055      ClosePicture;
000056      SetHeap(tempHz);   {restore normal heap}
000057      {$IFC fTrace}EP;{$ENDC}
000058  END;
000059
000060
000061  {$S DlgCold}
000062  PROCEDURE TPicObject.Free;
000063  BEGIN
000064     {$IFC fTrace}BP(7);{$ENDC}
000065     KillPicture(SELF.picture);
000066     SUPERSELF.Free;
000067     {$IFC fTrace}EP;{$ENDC}
000068  END;
000069
000070
000071  {$IFC fDebugMethods}
000072  {$S DlgDbg}
000073  PROCEDURE TPicObject.Fields(PROCEDURE Field(nameAndType: S255));
000074  BEGIN
000075      SUPERSELF.Fields(Field);
000076      Field('picHandle: LONGINT');
000077      Field('boxAtCreation: Rect');
000078      Field('');
000079  END;
000080  {$ENDC}
000081
000082
000083  {$S DlgCold}
000084  PROCEDURE TPicObject.Draw;
000085  VAR boxOnPad:   Rect;
000086  BEGIN
000087     {$IFC fTrace}BP(7);{$ENDC}
000088     thePad.LRectToRect(SELF.extentLRect, boxOnPad);
000089     DrawPicture(SELF.picture, boxOnPad);
000090     {$IFC fTrace}EP;{$ENDC}
000091  END;
```

```
000092
000093
000094  {$S DlgInit}
000095  END;
000096
000097
000098  METHODS OF TRectImage;
000099
000100
000101  {$S DlgAlloc}
000102  FUNCTION TRectImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000103      itsView: TView; itsPenState: PenState; withChildren: BOOLEAN): TRectImage;
000104  BEGIN
000105      {$IFC fTrace}BP(11);{$ENDC}
000106      IF object = NIL THEN
000107          object := NewObject(heap, THISCLASS);
000108      SELF := TRectImage(TDialogImage.CREATE(object, heap, itsExtent, itsId, itsView, withChildren));
000109
000110      SELF.penState := itsPenState;
000111      {$IFC fTrace}EP;{$ENDC}
000112  END;
000113
000114
000115  {$IFC fDebugMethods}
000116  {$S DlgDbg}
000117  PROCEDURE TRectImage.Fields(PROCEDURE Field(nameAndType: S255));
000118  BEGIN
000119      SUPERSELF.Fields(Field);
000120      Field(CONCAT('penState: RECORD pnLoc: Point; pnSize: Point; pnMode: INTEGER;',
000121                   'pnPat: ARRAY[0..7] OF Byte END'));     {actually a packed array--fix}
000122      Field('');
000123  END;
000124  {$ENDC}
000125
000126
000127  {$S DlgHot}
000128  PROCEDURE TRectImage.Draw;
000129  BEGIN
000130      {$IFC fTrace}BP(11);{$ENDC}
000131      SetPenState(SELF.penState);  {could first want to scale the pen size, via the Pad...}
000132      FrameLRect(SELF.extentLRect);
000133      {$IFC fTrace}EP;{$ENDC}
000134  END;
000135
000136
000137  {$S DlgLayout}
000138  FUNCTION  TRectImage.LaunchLayoutBox(view: TView): TImage;
000139      VAR newBox:  TLayoutBox;
```

```
000140  BEGIN
000141      {$IFC fTrace}BP(11);{$ENDC}
000142      newBox := TLayoutBox.CREATE(NIL, SELF.Heap, SELF.extentLRect, noID, NIL,
000143          view, SELF, stdThinBorders, TRUE, TRUE, FALSE);
000144      LaunchLayoutBox := newBox;
000145      {$IFC fTrace}EP;{$ENDC}
000146  END;
000147
000148
000149  {$S DlgInit}
000150  END;
000151
000152
000153  {-----------------------------------------------------------------------------------------------}
000154
000155
000156  METHODS OF TTextDialogImage;
000157
000158
000159  {$S DlgText}
000160  FUNCTION  TTextDialogImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
000161                itsView: TView; itsTypeStyle: TTypeStyle; itsInitialChars: S255): TTextDialogImage;
000162      VAR textimage:  TTextImage;
000163          editPara:   TEditPara;
000164          paraFormat: TParaFormat;
000165  BEGIN
000166      {$IFC fTrace}BP(11);{$ENDC}
000167      IF object = NIL THEN
000168          object := NewObject(heap, THISCLASS);
000169      SELF := TTextDialogImage(TImageWithID.CREATE(object, heap, itsExtent, itsId, itsView,  FALSE));
000170
000171      textImage := TTextImage.CREATE(NIL, heap, itsView, itsExtent,
000172           TText.CREATE(NIL, heap, TDialogView(itsView).styleSheet), TRUE);
000173
000174      textImage.text.txtImgList.InsLast(textImage);
000175
000176      paraFormat := TParaFormat.CREATE(NIL, heap, NIL);
000177      paraFormat.dfltTStyle := itsTypeStyle;
000178
000179      editPara := TEditPara.CREATE(NIL, heap, 0, paraFormat);
000180
000181      textImage.imageList.InsLast(textImage.NewParaImage(editPara, itsExtent, 0, 0));
000182      textImage.text.paragraphs.InsLast(editPara);
000183      editPara.ReplPString(0, editPara.size, @itsInitialChars);
000184
000185      SELF.textImage := textImage;
000186      textImage.RecomputeImages(actionNone, TRUE);
000187      SELF.wouldBeDraggable := TRUE;
```

```
000188      SELF.refCount := 1;
000189      {$IFC fTrace}EP;{$ENDC}
000190  END;
000191
000192
000193  {$S SgTxtRes}
000194  PROCEDURE TTextDialogImage.Free;
000195  BEGIN
000196      {$IFC fTrace}BP(11);{$ENDC}
000197      SELF.textImage.text.Free;
000198      SUPERSELF.Free;
000199      {$IFC fTrace}EP;{$ENDC}
000200  END;
000201
000202
000203  PROCEDURE TTextDialogImage.ChangeRefCountBy(delta: INTEGER);
000204  BEGIN
000205      {$IFC fTrace}BP(11);{$ENDC}
000206      SELF.refCount := SELF.refCount + delta;
000207      IF SELF.refCount <= 0 THEN
000208          SELF.Free;
000209      {$IFC fTrace}EP;{$ENDC}
000210  END;
000211
000212
000213  {$IFC fDebugMethods}
000214  {$S DlgDbg}
000215  PROCEDURE TTextDialogImage.Fields(PROCEDURE Field(nameAndType: S255));
000216  BEGIN
000217      SUPERSELF.Fields(Field);
000218      Field('textImage: TTextImage');
000219      Field('wouldBeDraggable: BOOLEAN');
000220      Field('refCount: INTEGER');
000221      Field('');
000222  END;
000223  {$ENDC}
000224
000225
000226  {$S DlgHot}
000227  FUNCTION TTextDialogImage.CursorAt(mouseLPt: LPoint): TCursorNumber;
000228  BEGIN
000229      {$IFC fTrace}BP(11);{$ENDC}
000230      IF SELF.Hit(mouseLPt) THEN
000231          CursorAt := textCursor
000232      ELSE {not mine}
000233          CursorAt := noCursor;
000234      {$IFC fTrace}EP;{$ENDC}
000235  END;
```

```
000236
000237
000238  {$S SgTxtRes}
000239  PROCEDURE TTextDialogImage.Draw;
000240  BEGIN
000241      {$IFC fTrace}BP(11);{$ENDC}
000242      SELF.textImage.Draw;
000243      {$IFC fTrace}EP;{$ENDC}
000244  END;
000245
000246
000247  {$S DlgLayout}
000248  FUNCTION  TTextDialogImage.LaunchLayoutBox(view: TView): TImage;
000249      VAR borders:        Rect;
000250          newBox:         TLayoutBox;
000251  BEGIN
000252      {$IFC fTrace}BP(11);{$ENDC}   {dubious--formerly intended uses in abeyance}
000253      IF SELF.wouldBeDraggable THEN
000254          borders := stdPlainBorders
000255      ELSE
000256          borders := zeroRect;
000257      newBox := TLayoutBox.CREATE(NIL, SELF.Heap, SELF.extentLRect, noID, NIL {parent},
000258          view, SELF, borders, FALSE, FALSE, FALSE);
000259      newBox.wouldMakeSelection := TRUE;
000260      newBox.suppressDrawingManipulee := FALSE;
000261      newBox.isDraggable := SELF.wouldBeDraggable;
000262      LaunchLayoutBox := newBox;
000263      {$IFC fTrace}EP;{$ENDC}
000264  END;
000265
000266
000267  {$S SgTxtRes}
000268  PROCEDURE TTextDialogImage.MousePress(mouseLPt: LPoint);
000269  BEGIN
000270      {$IFC fTrace}BP(11);{$ENDC}
000271      SELF.textImage.MousePress(mouseLPt);
000272      {$IFC fTrace}EP;{$ENDC}
000273  END;
000274
000275
000276  {$S SgTxtRes}
000277  PROCEDURE TTextDialogImage.OffsetBy(deltaLPt: LPoint);
000278  BEGIN
000279      {$IFC fTrace}BP(11);{$ENDC}
000280      SELF.textImage.OffsetBy(deltaLPt);
000281      SUPERSELF.OffsetBy(deltaLPt);
000282      {$IFC fTrace}EP;{$ENDC}
000283  END;
```

```
000284
000285
000286  {$S DlgInit}
000287  END;
000288
000289
000290  METHODS OF TFrameSelection;
000291
000292  {$S DlgHot}
000293  FUNCTION TFrameSelection.CREATE(object: TObject; heap: THeap;
000294          itsInputFrame: TInputFrame): TFrameSelection;
000295      VAR coSelection:    TSelection;
000296  BEGIN
000297      {$IFC fTrace}BP(11);{$ENDC}
000298      IF object = NIL THEN
000299          object := NewObject(heap, THISCLASS);
000300      SELF := TFrameSelection(TSelection.CREATE(object, heap, itsInputFrame.view,
000301                                                frameKind, zeroLPt));
000302
000303      SELF.inputFrame := itsInputFrame;
000304      SELF.boundLRect := itsInputFrame.textDialogImage.extentLRect;
000305      coSelection := itsInputFrame.view.NoSelection;  {put non-NIL coSelection}
000306      SELF.coSelection := coSelection;
000307      {$IFC fTrace}EP;{$ENDC}
000308  END;
000309
000310
000311  {$S DlgHot}
000312  FUNCTION  TFrameSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
000313  BEGIN
000314      {$IFC fTrace}BP(10);{$ENDC}
000315      CASE cmdNumber OF
000316          uModern, uClassic,u20Pitch, u15Pitch, u12Pitch, u10Pitch, u12Point, u14Point, u18Point, u24Point,
000317          uPlain, uBold, uItalic, uUnderline, uShadow, uOutline:
000318              CanDoCommand := FALSE;   {before coSelection could set to TRUE}
000319
000320          OTHERWISE
000321              CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
000322          END;
000323      {$IFC fTrace}EP;{$ENDC}
000324  END;
000325
000326
000327  {$S DlgHot}
000328  PROCEDURE TFrameSelection.KeyChar(ch: CHAR);
000329      VAR paraImage:      TParaImage;
000330          maxCharsString: S255;
000331  BEGIN
```

```
000332      {$IFC fTrace}BP(11);{$ENDC}
000333      paraImage := TParaImage(SELF.inputFrame.textDialogImage.textImage.imageList.First);
000334      IF (SELF.inputFrame.maxInputChars > paraImage.paragraph.size) OR
000335         (NOT InClass(SELF.coSelection, TInsertionPoint)) THEN  {can accept more}
000336          SELF.coSelection.KeyChar(ch)
000337      ELSE
000338          BEGIN
000339          IntToStr(SELF.inputFrame.maxInputChars, @maxCharsString);
000340          process.ArgAlert(1, maxCharsString);
000341          process.Stop(phTooManyChars);
000342          END;
000343      {$IFC fTrace}EP;{$ENDC}
000344  END;
000345
000346
000347  {$S DlgHot}
000348  PROCEDURE TFrameSelection.KeyEnter(dh, dv: INTEGER);
000349  BEGIN
000350      {$IFC fTrace}BP(11);{$ENDC}
000351      IF (dh <> 0) OR (dv <> 0) THEN
000352          SELF.KeyTab((dh < 0) OR (dv < 0));  {right and down keys are Forward}
000353      {$IFC fTrace}EP;{$ENDC}
000354  END;
000355
000356
000357  {$S DlgHot}
000358  PROCEDURE TFrameSelection.KeyReturn;
000359  BEGIN
000360      {$IFC fTrace}BP(11);{$ENDC}
000361      SELF.KeyTab(FALSE);
000362      {$IFC fTrace}EP;{$ENDC}
000363  END;
000364
000365
000366  {$S DlgHot}
000367  PROCEDURE TFrameSelection.KeyTab(fBackward: BOOLEAN);
000368      VAR dialogView:     TDialogView;
000369          dialogImage:    TDialogImage;
000370          s:              TListScanner;
000371          passedGo:       BOOLEAN;
000372          foundSuccessor: BOOLEAN;
000373          prevInputFrame: TInputFrame;
000374          nextInputFrame: TInputFrame;
000375          newFrameSel:    TFrameSelection;
000376          dialog:         TDialog;
000377  BEGIN
000378      {$IFC fTrace}BP(11);{$ENDC}
000379      dialogView := TDialogView(SELF.view);
```

```
000380      prevInputFrame := NIL;
000381      passedGo := FALSE;
000382      foundSuccessor := FALSE;
000383      dialog := TDialog(SELF.inputFrame.parent);
000384      s := dialog.children.Scanner;
000385      WHILE s.Scan(dialogImage) DO
000386          BEGIN
000387          IF dialogImage = SELF.inputFrame THEN  {found current frame}
000388              BEGIN
000389              IF not fBackward THEN
000390                  passedGo := TRUE
000391              ELSE {back-tab; use most recent input frame, if any}
000392                  BEGIN
000393                  IF prevInputFrame = NIL THEN  {there is none; can't do anything}
000394                      s.Done {with foundSuccessor still FALSE}
000395                  ELSE {found somebody!}
000396                      BEGIN
000397                      nextInputFrame := prevInputFrame;
000398                      foundSuccessor := TRUE;
000399                      END;
000400                  END;
000401              END;
000402          IF InClass(dialogImage, TInputFrame) THEN
000403              IF (passedGo AND (dialogImage <> SELF.inputFrame)) OR foundSuccessor THEN
000404                  BEGIN
000405                  IF passedGo THEN
000406                      nextInputFrame := TInputFrame(dialogImage); {else it's already set, if back-tab}
000407                  SELF.KeyPause;
000408
000409                  dialog.SelectInputFrame(nextInputFrame);
000410
000411                  foundSuccessor := TRUE;
000412                  s.Done;
000413                  END   {forward tabbing actually done}
000414              ELSE
000415                  prevInputFrame := TInputFrame(dialogImage);
000416          END; {scan of dialogImages}
000417      IF NOT foundSuccessor THEN
000418          SELF.CantDoIt;
000419      {ELSE
000420          SELF.window.CommitLast};
000421      {$IFC fTrace}EP;{$ENDC}
000422  END;
000423
000424
000425  {$S DlgHot}
000426  PROCEDURE TFrameSelection.MousePress(mouseLPt: LPoint);
000427  {called only if mouse comes BACK down inside alredy-selected Input Frame}
```

```
000428  BEGIN
000429      {$IFC fTrace}BP(11);{$ENDC}
000430      SUPERSELF.MousePress(mouseLPt);
000431      TDialogView(SELF.view).magnetCursor := textCursor;
000432      {$IFC fTrace}EP;{$ENDC}
000433  END;
000434
000435
000436  {$S DlgHot}
000437  PROCEDURE TFrameSelection.PerformCommand(command: TCommand; cmdPhase: TCmdPhase);
000438      VAR paragraph:      TParagraph;
000439          textImage:      TTextImage;
000440          paraImage:      TParaImage;
000441          noSelection:    TSelection;
000442  BEGIN
000443      {$IFC fTrace}BP(11);{$ENDC}
000444      textImage := SELF.inputFrame.textDialogImage.textImage;
000445      SUPERSELF.PerformCommand(command, cmdPhase);
000446      paraImage := TParaImage(textImage.imageList.First);
000447      paragraph := paraImage.paragraph;
000448      paragraph.NewStyle(0, paragraph.Size, SELF.inputFrame.inputTypeStyle);
000449      IF paragraph.size > SELF.inputFrame.maxInputChars THEN
000450          BEGIN {may need temp para here}
000451          paragraph.ReplPara(0, paragraph.size, paragraph, 0, SELF.inputFrame.maxInputChars);
000452          paraImage.changed := TRUE;
000453          paraImage.InvalLinesWith(0, MAXINT);
000454          textImage.RecomputeImages(actionNone, TRUE);
000455          SELF.window.CommitLast;
000456          noSelection := SELF.FreedAndReplacedBy(SELF.view.NoSelection);
000457          END;
000458      {$IFC fTrace}EP;{$ENDC}
000459  END;
000460
000461
000462  {$S DlgHot}
000463  PROCEDURE TFrameSelection.Restore;
000464  BEGIN
000465      {$IFC fTrace}BP(11);{$ENDC}
000466      TDialogView(SELF.view).currentDialogImage := SELF.inputFrame;
000467      SUPERSELF.Restore;
000468      {$IFC fTrace}EP;{$ENDC}
000469  END;
000470
000471
000472  {$S DlgInit}
000473  END;
000474
000475
```

```
000476  METHODS OF TPlannerView;
000477
000478  {$S SgLayout}
000479      FUNCTION  TPlannerView.CREATE(object: TObject; heap: THeap; itsViewBeingPlanned: TView;
000480                                    itsPanel: TPanel; itsAllowSketching: BOOLEAN;
000481                                    itsRetainPickedBox: BOOLEAN): TPlannerView;
000482          VAR newList:    TList;
000483      BEGIN
000484          {$IFC fTrace}BP(11);{$ENDC}
000485          IF object = NIL THEN
000486              object := NewObject(heap, THISCLASS);
000487          SELF := TPlannerView(TDialogView.CREATE(object, heap, itsViewBeingPlanned.extentLRect, itsPanel,
000488                      NIL, itsViewBeingPlanned.res));
000489
000490          WITH SELF DO
000491              BEGIN
000492              viewBeingPlanned := itsViewBeingPlanned;
000493              allowSketching := itsAllowSketching;
000494              retainPickedBox := itsRetainPickedBox;
000495              currentLayoutBox := NIL;
000496              END;
000497          {$IFC fTrace}EP;{$ENDC}
000498      END;
000499
000500
000501      {$IFC fDebugMethods}
000502  {$S DlgDbg}
000503      PROCEDURE TPlannerView.Fields(PROCEDURE Field(nameAndType: S255));
000504      BEGIN
000505          SUPERSELF.Fields(Field);
000506          Field('viewBeingPlanned: TView');
000507          Field('allowSketching: BOOLEAN');
000508          Field('retainPickedBox: BOOLEAN');
000509          Field('currentLayoutBox: TLayoutBox');
000510          Field('');
000511      END;
000512      {$ENDC}
000513
000514
000515  {$S SgLayout}
000516      FUNCTION  TPlannerView.CursorAt(mouseLPt: LPoint): TCursorNumber;
000517      VAR s:          TListScanner;
000518          layoutBox:  TLayoutBox;
000519          curCursor:  TCursorNumber;
000520      BEGIN
000521          {$IFC fTrace}BP(11);{$ENDC}
000522          IF SELF.mouseIsDown AND (SELF.magnetCursor <> noCursor) THEN
000523              CursorAt := SELF.magnetCursor
```

```
000524          ELSE
000525              BEGIN
000526              curCursor := noCursor;
000527              s := SELF.rootDialog.children.Scanner;
000528              WHILE s.Scan(layoutBox) DO
000529                  BEGIN
000530                  curCursor := layoutBox.CursorAt(mouseLPt);
000531                  IF curCursor <> noCursor THEN
000532                      s.Done
000533                  END;
000534              CursorAt := curCursor;
000535              END;
000536          {$IFC fTrace}EP;{$ENDC}
000537      END;
000538
000539
000540  {$S SgLayout}
000541      PROCEDURE TPlannerView.Draw;
000542          PROCEDURE DrawLayoutBox(obj: TObject);
000543          VAR layoutBox:    TLayoutBox;
000544          BEGIN
000545              layoutBox := TLayoutBox(obj);
000546              layoutBox.Draw;
000547          END;
000548      BEGIN
000549          {$IFC fTrace}BP(10);{$ENDC}
000550          SELF.EachActualPart(DrawLayoutBox);
000551          {$IFC fTrace}EP;{$ENDC}
000552      END;
000553
000554
000555  {$S SgLayout}
000556      PROCEDURE TPlannerView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
000557      BEGIN
000558          {$IFC fTrace}BP(11);{$ENDC}
000559           SELF.rootDialog.children.Each(DoToObject);
000560          {$IFC fTrace}EP;{$ENDC}
000561      END;
000562
000563
000564  {$S DlgAlloc}
000565      PROCEDURE TPlannerView.Init(itsListOfImages: TList);
000566          VAR s:                  TListScanner;
000567              t:                  TListScanner;
000568              image:              TImage;
000569              layoutBox:          TLayoutBox;
000570              otherLayoutBox:     TLayoutBox;
000571              nextButton:         TButton;
```

```
000572      BEGIN
000573          {$IFC fTrace}BP(10);{$ENDC}
000574          IF itsListOfImages <> NIL THEN
000575              BEGIN
000576              s := itsListOfImages.Scanner;       {create parallel structure}
000577              WHILE s.Scan(image) DO
000578                  BEGIN
000579                  layoutBox := SELF.NewLayoutBox(image);
000580                  IF layoutBox <> NIL THEN
000581                      SELF.rootDialog.AddImage(layoutBox); {it may well have its own children, already created}
000582                  END;
000583              END;
000584
000585          IF InClass(SELF.viewBeingPlanned, TDialogView) THEN {get buttonLayoutBoxes correctly entwined}
000586              BEGIN
000587              s := SELF.rootDialog.children.Scanner;
000588              WHILE s.Scan(layoutBox) DO
000589                  IF InClass(layoutBox, TButtonLayoutBox) THEN
000590                      BEGIN
000591                      nextButton := TButton(layoutBox.manipulee).nextSameSizedButton;
000592                      t := SELF.rootDialog.children.Scanner;
000593                      WHILE t.Scan(otherLayoutBox) DO
000594                          IF otherLayoutBox.manipulee = nextButton THEN {found it}
000595                              BEGIN
000596                              TButtonLayoutBox(layoutBox).nextSameSizedBox := TButtonLayoutBox(otherLayoutBox);
000597                              t.Done;
000598                              END;
000599                      END;
000600              END;
000601          {$IFC fTrace}EP;{$ENDC}
000602      END;
000603
000604
000605  {$S DlgAlloc}
000606      FUNCTION TPlannerView.NewLayoutBox(image: TImage): TLayoutBox;
000607      BEGIN
000608          {$IFC fTrace}BP(10);{$ENDC}
000609          NewLayoutBox := TLayoutBox(image.LaunchLayoutBox(SELF));
000610          {$IFC fTrace}EP;{$ENDC}
000611      END;
000612
000613
000614  {$S DlgLayout}
000615      PROCEDURE TPlannerView.MousePress(mouseLPt: LPoint);
000616      VAR panel:                  TPanel;
000617          layPickSelection:       TLayPickSelection;
000618          pickedBox:              TLayoutBox;
000619          s:                      TListScanner;
```

```
000620          layoutBox:              TLayoutBox;
000621          madeSelection:          BOOLEAN;
000622          editLegendSelection:    TEditLegendSelection;
000623              PROCEDURE InvrtOnThePad;
000624              BEGIN
000625                  pickedBox.Highlight(hOffToOn);
000626              END;
000627      BEGIN
000628          {$IFC fTrace}BP(10);{$ENDC}
000629          SELF.mouseIsDown := TRUE;
000630          SELF.magnetCursor := noCursor;
000631          panel := SELF.panel;
000632          madeSelection := FALSE;
000633          IF (panel.selection.kind <> nothingKind) AND (panel.selection.kind <> layPickKind) THEN
000634              BEGIN
000635              IF clickState.fShift THEN
000636                  madeSelection := TRUE
000637              ELSE
000638              IF InClass(panel.selection, TEditLegendSelection) THEN
000639                  BEGIN
000640                  editLegendSelection := TEditLegendSelection(panel.selection);
000641                  IF LPtInLRect(mouseLPt, editLegendSelection.legendLayoutBox.extentLRect) THEN
000642                      IF NOT LPtInLRect(mouseLPt, editLegendSelection.legendLayoutBox.titleTab.extentLRect) THEN
000643                          BEGIN
000644                          madeSelection := TRUE;
000645                          SELF.magnetCursor := textCursor;
000646                          END;
000647                  END;
000648              END;
000649
000650          IF madeSelection THEN
000651              panel.selection.MousePress(mouseLPt)
000652          ELSE
000653              BEGIN
000654              panel.BeginSelection;
000655              pickedBox := NIL;  {find who wants the mouse}
000656              s := SELF.rootDialog.children.Scanner;
000657              WHILE s.Scan(layoutBox) DO
000658                  BEGIN
000659                  layoutBox.ConsiderMouse(mouseLPt, madeSelection, pickedBox);
000660                  IF pickedBox <> NIL THEN
000661                      pickedBox.ComeForward;
000662                  IF madeSelection THEN
000663                      s.Done
000664                  ELSE
000665                  IF (pickedBox <> NIL)  THEN {got the title tab of somebody}
000666                      BEGIN
000667                      pickedBox.ChangeDragState(TRUE);
```

```
000668                      layPickSelection := TLayPickSelection(panel.selection.FreedAndReplacedBy(
000669                                  TLayPickSelection.CREATE(NIL, SELF.Heap, SELF, layPickKind, pickedBox,
000670                                  mouseLPt)));
000671                      panel.OnAllPadsDo(InvrtOnThePad);
000672                      SELF.magnetCursor := arrowCursor;
000673                      s.Done;
000674                      END
000675                  END;
000676  (*     NB: Here, when/if we allow sketching in layout, we would add code like:
000677
000678          IF pickedBox = NIL THEN
000679                  IF SELF.allowSketching THEN
000680                      LaySketchSelection := TLaySketchSelection(panel.selection.FreedAndReplacedBy(
000681                                  TLaySketchSelection.CREATE(NIL, SELF.Heap, SELF, mouseLPt)))   *)
000682              END;
000683          {$IFC fTrace}EP;{$ENDC}
000684      END;
000685
000686
000687  {$S DlgLayout}
000688      PROCEDURE TPlannerView.MouseMove(mouseLPt: LPoint);
000689      BEGIN
000690          {$IFC fTrace}BP(11);{$ENDC}
000691          TView.MouseMove(mouseLPt);  {do NOT do what TDialogView would do}
000692         {$IFC fTrace}EP;{$ENDC}
000693      END;
000694
000695
000696  {$S SgLayout}
000697      PROCEDURE TPlannerView.MouseRelease;
000698      BEGIN
000699          {$IFC fTrace}BP(11);{$ENDC}
000700          SELF.mouseIsDown := FALSE;
000701          SELF.magnetCursor := noCursor;
000702          TView.MouseRelease;         {do NOT do what TDialogView would do except for the above}
000703          {$IFC fTrace}EP;{$ENDC}
000704      END;
000705
000706
000707  {$S DlgInit}
000708  END;
000709
000710
000711  METHODS OF TLayoutBox;
000712
000713
000714  {$S SgLayout}
000715      FUNCTION TLayoutBox.CREATE(object: TObject; heap: THeap; baseExtent: LRect; itsID: S255;
```

```
000716                  itsParent: TLayoutBox; itsView: TView; itsManipulee: TImage; itsBorders: Rect;
000717                  itsResizable: BOOLEAN; itsSuppression: BOOLEAN; withChildren: BOOLEAN): TLayoutBox;
000718          VAR itsTitleTab:    TTitleTab;
000719              itsExtentLRect: LRect;
000720      BEGIN
000721          {$IFC fTrace}BP(11);{$ENDC}
000722          LRectAddBorders(baseExtent, itsBorders, itsExtentLRect);
000723          IF object = NIL THEN
000724              object := NewObject(heap, THISCLASS);
000725          SELF := TLayoutBox(TImageWithID.CREATE(object, heap, itsExtentLRect, itsID, itsView, withChildren));
000726
000727          IF itsBorders.top = 0 THEN
000728              itsTitleTab := SELF.NoTitleTab(SELF.Heap)
000729          ELSE
000730              itsTitleTab := TTitleTab.CREATE(NIL, heap, SELF,  - itsBorders.bottom - itsBorders.top + 1,
000731                                              itsID);
000732
000733          WITH SELF DO
000734              BEGIN
000735              parent := itsParent;
000736              titleTab := itsTitleTab;
000737              manipulee := itsManipulee;
000738              suppressDrawingManipulee := itsSuppression;
000739              wouldMakeSelection := FALSE;
000740              isResizable := itsResizable;
000741              isDraggable := TRUE;
000742              shouldFrame := TRUE;
000743              borders := itsBorders;
000744              hasDraggee := FALSE;
000745              END;
000746          {$IFC fTrace}EP;{$ENDC}
000747      END;
000748
000749
000750      PROCEDURE TLayoutBox.Free;
000751      BEGIN
000752          {$IFC fTrace}BP(10);{$ENDC}
000753          Free(SELF.titleTab);
000754          SUPERSELF.Free;
000755          {$IFC fTrace}EP;{$ENDC}
000756      END;
000757
000758
000759      {$IFC fDebugMethods}
000760      {$S DlgDbg}
000761      PROCEDURE TLayoutBox.Fields(PROCEDURE Field(nameAndType: S255));
000762      BEGIN
000763          SUPERSELF.Fields(Field);
```

```
000764          Field('manipulee: TImage');
000765          Field('titleTab: TTitleTab');
000766          Field('suppressDrawingManipulee: BOOLEAN');
000767          Field('isResizable: BOOLEAN');
000768          Field('borders: Rect');
000769          Field('wouldMakeSelection: BOOLEAN');
000770          Field('isDraggable: BOOLEAN');
000771          Field('shouldFrame: BOOLEAN');
000772          Field('hasDraggee: BOOLEAN');
000773          Field('');
000774      END;
000775      {$S SgLayout}
000776      {$ENDC}
000777
000778
000779      PROCEDURE TLayoutBox.ChangeDragState(enteringDrag: BOOLEAN);
000780      BEGIN
000781          {$IFC fTrace}BP(11);{$ENDC}
000782          SELF.hasDraggee := enteringDrag;
000783          IF SELF.parent <> NIL THEN
000784              IF InClass(SELF.parent, TLayoutBox) THEN
000785                  TLayoutBox(SELF.parent).ChangeDragState(enteringDrag);
000786          {$IFC fTrace}EP;{$ENDC}
000787      END;
000788
000789
000790      PROCEDURE  TLayoutBox.ConsiderMouse(mouseLPt: LPoint; VAR madeSelection: BOOLEAN;
000791              VAR pickedLayoutBox: TLayoutBox);
000792          VAR s:          TListScanner;
000793              layoutBox:  TLayoutBox;
000794      BEGIN
000795          {$IFC fTrace}BP(11);{$ENDC}
000796          pickedLayoutBox := NIL;
000797          madeSelection := FALSE;
000798          IF NOT SELF.Hit(mouseLPt) THEN
000799              {it ain't me}
000800          ELSE
000801              BEGIN
000802              IF LRectHasLPt(SELF.titleTab.extentLRect, mouseLPt) THEN
000803                  BEGIN
000804                  pickedLayoutBox := SELF;
000805                  SELF.TabGrabbed;        {so that page status dialog can react now}
000806                  END
000807              ELSE
000808              IF SELF.wouldMakeSelection THEN
000809                  BEGIN
000810                  madeSelection := TRUE;
000811                  pickedLayoutBox := SELF;
```

```
000812                  SELF.MousePress(mouseLPt);
000813                  END
000814              ELSE    {not my title tab, and I myself don't make selections; how about it, kids?}
000815                  BEGIN
000816                  IF SELF.children <> NIL THEN
000817                      BEGIN
000818                      s := SELF.children.Scanner;
000819                      WHILE s.Scan(layoutBox) DO
000820                          BEGIN
000821                          layoutBox.ConsiderMouse(mouseLPt, madeSelection, pickedLayoutBox);
000822                          IF madeSelection OR (pickedLayoutBox <> NIL) THEN
000823                              s.Done;
000824                          END;
000825                      END;
000826                  END;
000827              END;
000828          {$IFC fTrace}EP;{$ENDC}
000829      END;
000830
000831
000832      FUNCTION TLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber;
000833      VAR s:          TListScanner;
000834          layoutBox:  TLayoutBox;
000835          curCursor:  TCursorNumber;
000836      BEGIN
000837          {$IFC fTrace}BP(11);{$ENDC}
000838          curCursor := noCursor;
000839          IF SELF.Hit(mouseLPt) THEN
000840              BEGIN
000841              IF SELF.titleTab <> NIL THEN
000842                  IF SELF.titleTab.Hit(mouseLPt) THEN
000843                      curCursor := arrowCursor;
000844
000845              IF curCursor = noCursor THEN
000846                  IF SELF.children <> NIL THEN
000847                      BEGIN
000848                      s := SELF.children.Scanner;
000849                      WHILE s.Scan(layoutBox) DO
000850                          BEGIN
000851                          curCursor := layoutBox.CursorAt(mouseLPt);
000852                          IF curCursor <> noCursor THEN
000853                              s.Done;
000854                          END;
000855                      END;
000856              END;
000857          CursorAt := curCursor;
000858          {$IFC fTrace}EP;{$ENDC}
000859      END;
```

```
000860
000861
000862      PROCEDURE TLayoutBox.Draw;
000863          PROCEDURE YouDraw(obj: TObject);
000864              VAR dialogImage: TDialogImage;
000865          BEGIN
000866              dialogImage := TDialogImage(obj);
000867              IF dialogImage.isActive THEN
000868                  dialogImage.Draw;
000869          END;
000870      BEGIN
000871          {$IFC fTrace}BP(11);{$ENDC}
000872          IF LRectIsVisible(SELF.extentLRect) OR SELF.hasDraggee THEN
000873              BEGIN
000874              SELF.EachActualPart(YouDraw);
000875              SELF.DrawJustMe;
000876              END;
000877          {$IFC fTrace}EP;{$ENDC}
000878      END;
000879
000880
000881      PROCEDURE TLayoutBox.DrawJustMe;
000882          VAR titleTab:   TTitleTab;
000883      BEGIN
000884          {$IFC fTrace}BP(10);{$ENDC}
000885          IF LRectIsVisible(SELF.extentLRect) THEN
000886              BEGIN
000887              titleTab := SELF.titleTab;
000888              IF titleTab <> NIL THEN  {currently every layout box MUST have a title tab, so this is
000889                                        unnecessary}
000890                  IF NOT EmptyLRect(titleTab.extentLRect) THEN
000891                      titleTab.Draw;
000892
000893              IF NOT SELF.suppressDrawingManipulee THEN
000894                  IF SELF.manipulee <> NIL THEN
000895                      SELF.manipulee.Draw;
000896
000897              PenNormal;
000898              IF SELF.IsDraggable AND SELF.shouldFrame THEN
000899              FrameLRect(SELF.extentLRect);  {draw overall box}
000900              END;
000901          {$IFC fTrace}EP;{$ENDC}
000902      END;
000903
000904
000905      PROCEDURE TLayoutBox.FreeManipulee;
000906      BEGIN
000907          {$IFC fTrace}BP(10);{$ENDC}
```

```
000908          Free(SELF.manipulee);
000909          SELF.manipulee := NIL;
000910          {$IFC fTrace}EP;{$ENDC}
000911      END;
000912
000913
000914      PROCEDURE TLayoutBox.Highlight(highTransit: THighTransit);
000915      BEGIN
000916          {$IFC fTrace}BP(10);{$ENDC}
000917          IF (SELF.titleTab <> NIL) THEN
000918              BEGIN
000919              InvrtLRect(SELF.titleTab.extentLRect);
000920              PenNormal;
000921              FrameLRect(SELF.titleTab.extentLRect);
000922              END;
000923          {$IFC fTrace}EP;{$ENDC}
000924      END;
000925
000926
000927      PROCEDURE TLayoutBox.MousePress(mouseLPT: LPoint);
000928          VAR layoutBox:  TLayoutBox;
000929              s:          TListScanner;
000930      BEGIN
000931          {$IFC fTrace}BP(11);{$ENDC}
000932          IF SELF.children <> NIL THEN
000933              BEGIN
000934              s := SELF.children.Scanner;
000935              WHILE s.Scan(layoutBox) DO
000936                  IF layoutBox.DownAt(mouseLPt) <> NIL THEN
000937                      s.Done;
000938              END;
000939          {$IFC fTrace}EP;{$ENDC}
000940      END;
000941
000942
000943      PROCEDURE TLayoutBox.Move(deltaLPt: LPoint);
000944      VAR oldLRect:   LRect;
000945          newLRect:   LRect;
000946          heading:    THeading;
000947              PROCEDURE InvalOnThePad;
000948              BEGIN
000949                  thePad.InvalLRect(oldLRect);
000950                  thePad.InvalLRect(newLRect);
000951              END;
000952      BEGIN
000953          {$IFC fTrace}BP(10);{$ENDC}
000954          oldLRect := SELF.extentLRect;
000955          SELF.OffsetBy(deltaLPt);
```

```
000956          newLRect := SELF.extentLRect;
000957          SELF.view.panel.OnAllPadsDo(InvalOnThePad);
000958          {$IFC fTrace}EP;{$ENDC}
000959      END;
000960
000961
000962  {$S DlgDbg}
000963      FUNCTION TLayoutBox.NoTitleTab(heap: THeap): TTitleTab;
000964      BEGIN
000965          {$IFC fTrace}BP(11);{$ENDC}
000966          NoTitleTab := TTitleTab.CREATE(NIL, heap, SELF, 0, noID);
000967          {$IFC fTrace}EP;{$ENDC}
000968      END;
000969  {$S SgLayout}
000970
000971
000972      PROCEDURE TLayoutBox.OffsetBy(deltaLPt: LPoint);
000973      BEGIN
000974          {$IFC fTrace}BP(10);{$ENDC}
000975          IF SELF.manipulee <> NIL THEN
000976              SELF.manipulee.OffsetBy(deltaLPt);   {offset MY manipulee, but not my children's, since my
000977                                                    my manipulee's OffsetBy will have done that already}
000978          SELF.OffsetLayoutBoxBy(deltaLPt, TRUE);
000979          {$IFC fTrace}EP;{$ENDC}
000980      END;
000981
000982
000983      PROCEDURE TLayoutBox.OffsetLayoutBoxBy(deltaLPt: LPoint; textImageAsWell: BOOLEAN);
000984                                             {does NOT offset manipulee}
000985          PROCEDURE YouOffset(obj: TObject);
000986              VAR layoutBox: TLayoutBox;
000987          BEGIN
000988              layoutBox := TLayoutBox(obj);
000989              layoutBox.OffsetLayoutBoxBy(deltaLPt, textImageAsWell);
000990          END;
000991      BEGIN
000992          {$IFC fTrace}BP(11);{$ENDC}
000993          IF SELF.titleTab <> NIL THEN
000994              SELF.titleTab.OffsetBy(deltaLPt);
000995    {$H-} OffsetLRect(SELF.extentLRect, deltaLPt.h, deltaLPt.v); {$H+}
000996          SELF.EachActualPart(YouOffset);  {tells children}
000997          {$IFC fTrace}EP;{$ENDC}
000998      END;
000999
001000
001001      PROCEDURE TLayoutBox.RecalcExtent;
001002          VAR s:              TListScanner;
001003              newExtent:      LRect;
```

```
001004              layoutBox:      TLayoutBox;
001005              oldExtent:      LRect;
001006          PROCEDURE InvalOldAndNew;
001007              BEGIN
001008              thePad.InvalLRect(oldExtent);
001009              thePad.InvalLRect(newExtent);
001010              END;
001011      BEGIN
001012          {$IFC fTrace}BP(3);{$ENDC}
001013          oldExtent := SELF.extentLRect;
001014          newExtent := SELF.manipulee.extentLRect;
001015
001016          IF SELF.children <> NIL THEN
001017              IF SELF.children.Size > 0 THEN
001018                  BEGIN
001019                  newExtent := zeroLRect;
001020                  s := SELF.children.Scanner;
001021                  WHILE s.Scan(layoutBox) DO
001022                      IF EmptyLRect(newExtent) THEN
001023                          newExtent := layoutBox.extentLRect
001024                      ELSE
001025                          UnionLRect(newExtent, layoutBox.extentLRect, newExtent);
001026                  END;
001027
001028          LRectAddBorders(newExtent, SELF.borders, newExtent);
001029          IF NOT equalLRect(oldExtent, newExtent) THEN
001030              BEGIN
001031              SELF.Resize(newExtent);
001032              SELF.view.panel.OnAllPadsDo(InvalOldAndNew);
001033              END;
001034          IF SELF.parent <> NIL THEN
001035              SELF.parent.RecalcExtent;
001036          {$IFC fTrace}EP;{$ENDC}
001037      END;
001038
001039
001040      PROCEDURE TLayoutBox.Resize(newExtent: LRect);
001041          VAR newTitleExtent: LRect;
001042              titleTab:       TTitleTab;
001043      BEGIN
001044          {$IFC fTrace}BP(3);{$ENDC}
001045          titleTab := SELF.titleTab;
001046
001047          IF titleTab <> NIL THEN
001048              BEGIN
001049              newTitleExtent := newExtent;
001050              newTitleExtent.bottom := newTitleExtent.top +
001051                   (titleTab.extentLRect.bottom - titleTab.extentLRect.top); {i.e., preserve old height}
```

```
001052              titleTab.Resize(newTitleExtent);
001053              END;
001054          SUPERSELF.Resize(newExtentLRect);
001055          {$IFC fTrace}EP;{$ENDC}
001056      END;
001057
001058
001059      PROCEDURE TLayoutBox.TabGrabbed;
001060      BEGIN
001061          {$IFC fTrace}BP(10);{$ENDC}
001062          {$IFC fTrace}EP;{$ENDC}
001063      END;
001064
001065
001066  {$S DlgInit}
001067  END;
001068
001069
001070  METHODS OF TLegendLayoutBox;
001071
001072
001073  {$S SgLayout}
001074      FUNCTION  TLegendLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView; itsLegend: TLegend
001075              ): TLegendLayoutBox;
001076      VAR itsTitleTab:    TTitleTab;
001077          itsExtentLRect: LRect;
001078          myBorders:      Rect;
001079      BEGIN
001080          {$IFC fTrace}BP(11);{$ENDC}
001081          IF itsLegend.wouldBeDraggable THEN
001082              myBorders := stdPlainBorders
001083          ELSE
001084              myBorders := zeroRect;
001085          IF object = NIL THEN
001086              object := NewObject(heap, THISCLASS);
001087          SELF := TLegendLayoutBox(TLayoutBox.CREATE(object, heap, itsLegend.extentLRect, noID, NIL,
001088              itsView, itsLegend, myBorders, FALSE, FALSE, FALSE));
001089
001090          WITH SELF DO
001091              BEGIN
001092              isDraggable := itsLegend.wouldBeDraggable;
001093              shouldFrame := itsLegend.wouldBeDraggable;
001094              textDialogImage := NIL;
001095              wouldMakeSelection := TRUE;   {client could override, somehow?}
001096              END;
001097          {$IFC fTrace}EP;{$ENDC}
001098      END;
001099
```

```
001100
001101      {$IFC fDebugMethods}
001102      {$S DlgDbg}
001103      PROCEDURE TLegendLayoutBox.Fields(PROCEDURE Field(nameAndType: S255));
001104      BEGIN
001105          SUPERSELF.Fields(Field);
001106          Field('textDialogImage: TTextDialogImage');
001107          Field('');
001108      END;
001109      {$S SgLayout}
001110      {$ENDC}
001111
001112
001113      FUNCTION TLegendLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber;
001114      VAR  curCursor:  TCursorNumber;
001115      BEGIN
001116          {$IFC fTrace}BP(11);{$ENDC}
001117          curCursor := noCursor;
001118          IF SELF.Hit(mouseLPt) THEN
001119              IF SELF.titleTab.Hit(mouseLPt) THEN
001120                  curCursor := arrowCursor
001121              ELSE
001122              IF SELF.wouldMakeSelection THEN
001123                  curCursor := textCursor;
001124          CursorAt := curCursor;
001125          {$IFC fTrace}EP;{$ENDC}
001126      END;
001127
001128
001129      PROCEDURE TLegendLayoutBox.Draw;
001130          VAR titleTab:           TTitleTab;
001131      BEGIN
001132          {$IFC fTrace}BP(10);{$ENDC}
001133          IF LRectIsVisible(SELF.extentLRect) THEN
001134              BEGIN
001135              IF SELF.isDraggable THEN
001136                  SELF.titleTab.Draw;
001137
001138              IF SELF.textDialogImage <> NIL THEN {+SW+}
001139                  SELF.textDialogImage.Draw
001140              ELSE
001141                  SELF.manipulee.Draw;
001142
001143              PenNormal;
001144              IF SELF.IsDraggable AND SELF.shouldFrame THEN
001145                  FrameLRect(SELF.extentLRect);  {draw overall box}
001146              END;
001147          {$IFC fTrace}EP;{$ENDC}
```

```
001148      END;
001149
001150
001151      PROCEDURE TLegendLayoutBox.MousePress(mouseLPT: LPoint);
001152          VAR editLegendSelection:    TEditLegendSelection;
001153      BEGIN
001154          {$IFC fTrace}BP(11);{$ENDC}
001155          TPlannerView(SELF.view).magnetCursor := textCursor;
001156          LRectHaveLPt(SELF.manipulee.extentLRect, mouseLPt);
001157          editLegendSelection := TEditLegendSelection(SELF.view.panel.selection.FreedAndReplacedBy(
001158                                              TEditLegendSelection.CREATE(NIL, SELF.Heap, SELF, mouseLPT)));
001159          editLegendSelection.textDialogImage.MousePress(mouseLPt);
001160          {$IFC fTrace}EP;{$ENDC}
001161      END;
001162
001163
001164      PROCEDURE TLegendLayoutBox.OffsetLayoutBoxBy(deltaLPt: LPoint; textImageAsWell: BOOLEAN);
001165                                          {does NOT offset manipulee}
001166      BEGIN
001167          {$IFC fTrace}BP(11);{$ENDC}
001168          SUPERSELF.OffsetLayoutBoxBy(deltaLPt, textImageAsWell);
001169          IF NOT textImageAsWell THEN
001170              deltaLPt.v := 0; {don't do it vertically}
001171          IF SELF.textDialogImage <> NIL THEN
001172              SELF.textDialogImage.OffsetBy(deltaLPt);
001173          {$IFC fTrace}EP;{$ENDC}
001174      END;
001175
001176
001177      PROCEDURE TLegendLayoutBox.RecalcExtent;
001178          VAR newExtent:          LRect;
001179              oldExtent:          LRect;
001180              paraImage:          TParaImage;
001181              textDialogImage:    TTextDialogImage;
001182              legPara:            TParagraph;
001183          PROCEDURE InvalOldAndNew;
001184              BEGIN
001185              thePad.InvalLRect(oldExtent);
001186              thePad.InvalLRect(newExtent);
001187              END;
001188          PROCEDURE PourItBack(obj: TObject);
001189              VAR paragraph:  TParagraph;
001190          BEGIN
001191              paragraph := TParaImage(obj).paragraph;
001192              legPara.ReplPara(0, legPara.size, paragraph, 0, paragraph.size);
001193          END;
001194      BEGIN
001195          {$IFC fTrace}BP(11);{$ENDC}
```

```
001196          textDialogImage := SELF.textDialogImage;
001197          IF textDialogImage <> NIL THEN
001198              BEGIN
001199              paraImage := TParaImage(textDialogImage.textImage.imageList.First);
001200              legPara := TLegend(SELF.manipulee).paragraph;
001201              textDialogImage.textImage.FilterAndDo(paraImage, PourItBack);
001202              END;
001203          TLegend(SELF.manipulee).RecalcExtent;
001204
001205          oldExtent := SELF.extentLRect;
001206          newExtent := SELF.manipulee.extentLRect;
001207
001208          LRectAddBorders(newExtent, SELF.borders, newExtent);
001209          IF NOT equalLRect(oldExtent, newExtent) THEN
001210              BEGIN
001211              SELF.Resize(newExtent);
001212              SELF.view.panel.OnAllPadsDo(InvalOldAndNew);
001213              END;
001214
001215          IF SELF.parent <> NIL THEN
001216              SELF.parent.RecalcExtent;
001217          {$IFC fTrace}EP;{$ENDC}
001218      END;
001219
001220
001221  {$S DlgInit}
001222  END;
001223
001224
001225  METHODS OF TButtonLayoutBox;
001226
001227
001228  {$S DlgLayout}
001229      FUNCTION TButtonLayoutBox.CREATE(object: TObject; heap: THeap; itsButton: TButton;
001230                                       itsView: TView): TButtonLayoutBox;
001231      BEGIN
001232          {$IFC fTrace}BP(11);{$ENDC}
001233          IF object = NIL THEN
001234              object := NewObject(heap, THISCLASS);
001235          SELF := TButtonLayoutBox(TLayoutBox.CREATE(object, heap, itsButton.extentLRect, itsButton.id, NIL,
001236                                                     itsView, itsButton, stdIDBorders, FALSE, TRUE, TRUE));
001237
001238          SELF.nextSameSizedBox := SELF; {will be correctly set by TPlannerView.CREATE after all layout
001239                                         boxes made}
001240          SELF.oldLegendTopLeft := itsButton.legend.extentLRect.topLeft;
001241          SELF.AddImage(TDialogImage(itsButton.legend.LaunchLayoutBox(itsView)));
001242          {$IFC fTrace}EP;{$ENDC}
001243      END;
```

```
001244
001245
001246      {$IFC fDebugMethods}
001247      {$S DlgDbg}
001248      PROCEDURE TButtonLayoutBox.Fields(PROCEDURE Field(nameAndType: S255));
001249      BEGIN
001250          SUPERSELF.Fields(Field);
001251          Field('nextSameSizedBox: TButtonLayoutBox');
001252          Field('oldLegendTopLeft: LPoint');
001253          Field('');
001254      END;
001255      {$ENDC}
001256
001257
001258  {$S DlgLayout}
001259      PROCEDURE  TButtonLayoutBox.ConsiderMouse(mouseLPt: LPoint; VAR madeSelection: BOOLEAN;
001260              VAR pickedLayoutBox: TLayoutBox);
001261      BEGIN
001262          {$IFC fTrace}BP(11);{$ENDC}
001263          pickedLayoutBox := NIL;
001264          madeSelection := FALSE;
001265          IF SELF.Hit(mouseLPt) THEN
001266              BEGIN
001267              pickedLayoutBox := SELF;
001268              IF NOT LRectHasLPt(SELF.titleTab.extentLRect, mouseLPt) THEN
001269              {hit on interior -- hence, editing button text}
001270                  BEGIN
001271                  madeSelection := TRUE;
001272                  LRectHaveLPt(TLayoutBox(SELF.children.First).extentLRect, mouseLPt);
001273                  SELF.MousePress(mouseLPt);
001274                  END;
001275              END;
001276          {$IFC fTrace}EP;{$ENDC}
001277      END;
001278
001279
001280  {$S DlgLayout}
001281      FUNCTION  TButtonLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber;
001282      BEGIN
001283          {$IFC fTrace}BP(11);{$ENDC}
001284          IF NOT SELF.Hit(mouseLPt) THEN
001285              CursorAt := noCursor
001286          ELSE
001287          IF SELF.titleTab.Hit(mouseLPt) THEN
001288              CursorAt := arrowCursor
001289          ELSE
001290              CursorAt := textCursor;
001291          {$IFC fTrace}EP;{$ENDC}
```

```
001292      END;
001293
001294
001295  {$S DlgLayout}
001296      PROCEDURE  TButtonLayoutBox.DrawJustMe;
001297          VAR s:          TListScanner;
001298              layoutBox:  TLayoutBox;
001299      BEGIN
001300          {$IFC fTrace}BP(11);{$ENDC}
001301          IF LRectIsVisible(SELF.extentLRect) THEN
001302              BEGIN
001303              SELF.titleTab.Draw;
001304              TButton(SELF.manipulee).DrawJustMe;  {draws just the roundRect; my child will draw the text}
001305              PenNormal;
001306              FrameLRect(SELF.extentLRect);  {draw overall box}
001307              END;
001308          {$IFC fTrace}EP;{$ENDC}
001309      END;
001310
001311
001312  {$S SgLayout}
001313      PROCEDURE TButtonLayoutBox.OffsetBy(deltaLPt: LPoint);
001314      BEGIN
001315          {$IFC fTrace}BP(11);{$ENDC}
001316          SUPERSELF.OffsetBy(deltaLPt);
001317    {$H-} LPtPlusLPt(SELF.oldLegendTopLeft, deltaLPt, SELF.oldLegendTopLeft); {$H+}
001318          {$IFC fTrace}EP;{$ENDC}
001319      END;
001320
001321
001322  {$S DlgLayout}
001323      PROCEDURE TButtonLayoutBox.RecalcExtent;
001324          VAR nextBox:            TButtonLayoutBox;
001325              oldLegendTopLeft:  LPoint;
001326              newLegendTopLeft:  LPoint;
001327              deltaLPt:           LPoint;
001328              legendBox:          TLegendLayoutBox;
001329      BEGIN
001330          {$IFC fTrace}BP(11);{$ENDC}
001331          legendBox := TLegendLayoutBox(SELF.children.First);
001332          newLegendTopLeft := legendBox.manipulee.extentLRect.topLeft;
001333          oldLegendTopLeft := SELF.oldLegendTopLeft;
001334          IF NOT EqualLPt(oldLegendTopLeft, newLegendTopLeft) THEN
001335              BEGIN
001336              LPtMinusLPt(newLegendTopLeft, oldLegendTopLeft, deltaLPt);
001337              legendBox.OffsetLayoutBoxBy(deltaLPt, TRUE);
001338              SELF.oldLegendTopLeft := newLegendTopLeft;
001339              END;
```

```
001340
001341          nextBox := SELF;
001342          REPEAT
001343              nextBox.RecalcJustMe;
001344              nextBox := nextBox.nextSameSizedBox;
001345          UNTIL
001346              nextBox = SELF;
001347
001348          IF SELF.parent <> NIL THEN
001349              SELF.parent.RecalcExtent;
001350          {$IFC fTrace}EP;{$ENDC}
001351      END;
001352
001353
001354  {$S DlgLayout}
001355      PROCEDURE TButtonLayoutBox.RecalcJustMe;  {my manipulee's size may've changed}
001356          VAR nextBox:    TButtonLayoutBox;
001357              oldExtent:  LRect;
001358              newExtent:  LRect;
001359          PROCEDURE InvalOldAndNew;
001360              BEGIN
001361              thePad.InvalLRect(oldExtent);
001362              thePad.InvalLRect(newExtent);
001363              END;
001364      BEGIN
001365          {$IFC fTrace}BP(11);{$ENDC}
001366          oldExtent := SELF.extentLRect;
001367          newExtent := SELF.manipulee.extentLRect;
001368          LRectAddBorders(newExtent, SELF.borders, newExtent);
001369          SELF.Resize(newExtent);
001370          SELF.view.panel.OnAllPadsDo(InvalOldAndNew);
001371          {$IFC fTrace}EP;{$ENDC}
001372      END;
001373
001374
001375  {$S DlgInit}
001376  END;
001377
001378
001379  METHODS OF TTitleTab;
001380
001381  {$S SgLayout}
001382      FUNCTION  TTitleTab.CREATE(object: TObject; heap: THeap; itsLayoutBox: TLayoutBox; itsHeight: INTEGER;
001383          itsCaption: S255): TTitleTab;
001384          VAR extentLRect:    LRect;
001385              legend:         TLegend;
001386              location:       LPoint;
001387  {$IFC libraryVersion <= 20}   {* * * P E P S I * * *}
```

```
001388              fInfo:          TFInfo;
001389  {$ELSEC}                      {* *  S P R I N G  * *}
001390              fInfo:          FontInfo;
001391  {$ENDC}
001392              width:          INTEGER;
001393              newLegTopLeft:  LPoint;
001394      BEGIN
001395          {$IFC fTrace}BP(11);{$ENDC}
001396          WITH itsLayoutBox.extentLRect DO
001397    {$H-}     SetLRect(extentLRect, left, top, right, top + itsHeight);          {$H+}
001398
001399          IF object = NIL THEN
001400              object := NewObject(heap, THISCLASS);
001401          SELF := TTitleTab(TImage.CREATE(object, heap, extentLRect, itsLayoutBox.view));
001402
001403          SELF.layoutBox := itsLayoutBox;
001404
001405          IF itsCaption <> '' THEN {need to create a TLegend object for it}
001406              BEGIN
001407              legend := TLegend.CREATE(NIL, SELF.Heap, itsCaption, SELF.view, zeroLPt, titleTypeStyle);
001408              WITH itsLayoutBox.extentLRect DO
001409    {$H-}         SetLPt(newLegTopLeft, (right + left - legend.extentLRect.right) DIV 2,
001410                                        top + (SELF.view.res.v DIV 30));
001411    {$H+}     legend.OffsetTo(newLegTopLeft);
001412
001413              SELF.legend := legend;
001414              WITH legend.extentLRect DO
001415                  IF right - left > SELF.extentLRect.right - SELF.extentLRect.left THEN
001416                      SELF.shouldDrawLegend := FALSE
001417                  ELSE
001418                      SELF.shouldDrawLegend := TRUE;
001419              END
001420          ELSE
001421              BEGIN
001422              SELF.legend := NIL;
001423              SELF.shouldDrawLegend := FALSE;
001424              END;
001425          {$IFC fTrace}EP;{$ENDC}
001426      END;
001427
001428
001429  {$S SgLayout}
001430      PROCEDURE TTitleTab.Free;
001431      BEGIN
001432          {$IFC fTrace}BP(11);{$ENDC}
001433          Free(SELF.legend);
001434          SUPERSELF.Free;
001435          {$IFC fTrace}EP;{$ENDC}
```

```
001436      END;
001437
001438
001439      {$IFC fDebugMethods}
001440  {$S DlgDbg}
001441      PROCEDURE TTitleTab.Fields(PROCEDURE Field(nameAndType: S255));
001442      BEGIN
001443          SUPERSELF.Fields(Field);
001444          Field('layoutBox: TLayoutBox');
001445          Field('legend: TLegend');
001446          Field('shouldDrawLegend: BOOLEAN');
001447          Field('');
001448      END;
001449      {$S SgLayout}
001450      {$ENDC}
001451
001452
001453  {$S SgLayout}
001454      PROCEDURE TTitleTab.Draw;
001455      BEGIN
001456          {$IFC fTrace}BP(11);{$ENDC}
001457          PenNormal;
001458          FillLRect(SELF.extentLRect, lPatWhite);
001459          FrameLRect(SELF.extentLRect);
001460          IF SELF.shouldDrawLegend THEN      {it exists and is small enough to fit}
001461              SELF.legend.Draw;
001462          {$IFC fTrace}EP;{$ENDC}
001463      END;
001464
001465
001466  {$S SgLayout}
001467      PROCEDURE TTitleTab.OffsetBy(deltaLPt: LPoint);
001468      BEGIN
001469          {$IFC fTrace}BP(11);{$ENDC}
001470          IF SELF.legend <> NIL THEN
001471              SELF.legend.OffsetBy(deltaLPt);
001472          SUPERSELF.OffsetBy(deltaLPt);
001473          {$IFC fTrace}EP;{$ENDC}
001474      END;
001475
001476
001477  {$S SgLayout}
001478      PROCEDURE TTitleTab.Resize{(newExtent: LRect)};
001479          VAR myCaption:      S255;
001480  {$IFC libraryVersion <= 20}   {* * * P E P S I * * *}
001481              fInfo:          TFInfo;
001482  {$ELSEC}                      {* *  S P R I N G  * *}
001483              fInfo:          FontInfo;
```

```
001484  {$ENDC}
001485              strLocation:    LPoint;
001486              captionWidth:   INTEGER;
001487              deltaLPt:       LPoint;
001488              typeStyle:      TTypeStyle;
001489      BEGIN
001490          {$IFC fTrace}BP(11);{$ENDC}  {this does the wrong thing for high view resolutions; must fix}
001491          IF SELF.legend <> NIL THEN
001492              BEGIN
001493              SELF.legend.GetString(myCaption);
001494              SELF.legend.paragraph.StyleAt(0, typeStyle);
001495              SetQDTypeStyle(typeStyle);
001496              GetFontInfo(fInfo);
001497              captionWidth := StringWidth(myCaption);
001498      {$H-}   WITH newExtentLRect, fInfo DO
001499                  SetLPt(strLocation, ((left + right - captionWidth) DIV 2),
001500                              bottom - descent);      {had had a -2 here}
001501                  {$H+}
001502              SetLPt(deltaLPt, strLocation.h - SELF.legend.location.h,
001503                                  strLocation.v - SELF.legend.location.v);
001504              SELF.legend.OffsetBy(deltaLPt); {do more cleverly -- maybe TLegend.OffsetTo}
001505              WITH SELF.legend.extentLRect DO
001506                  IF right - left > newExtent.right - newExtent.left THEN
001507                      SELF.shouldDrawLegend := FALSE
001508                  ELSE
001509                      SELF.shouldDrawLegend := TRUE;
001510              END;
001511          SELF.extentLRect := newExtentLRect;
001512          {$IFC fTrace}EP;{$ENDC}
001513      END;
001514
001515
001516  {$S DlgInit}
001517  END;
001518
001519
001520  METHODS OF TLayPickSelection;
001521
001522
001523  {$S SgLayout}
001524      FUNCTION  TLayPickSelection.CREATE(object: TObject; heap: THeap; itsView: TPlannerView; itsKind: INTEGER;
001525                        itsLayoutBox: TLayoutBox; itsAnchorLPt: LPoint): TLayPickSelection;
001526      BEGIN
001527          {$IFC fTrace}BP(11);{$ENDC}
001528          IF object = NIL THEN
001529              object := NewObject(heap, THISCLASS);
001530          SELF := TLayPickSelection(TSelection.CREATE(object, heap, itsView, itsKind, itsAnchorLPt));
001531
```

```
001532          SELF.layoutBox := itsLayoutBox;
001533          SELF.boundLRect := itsLayoutBox.extentLRect;
001534          itsView.currentLayoutBox := itsLayoutBox;
001535          {$IFC fTrace}EP;{$ENDC}
001536      END;
001537
001538
001539      {$IFC fDebugMethods}
001540  {$S DlgDbg}
001541      PROCEDURE TLayPickSelection.Fields(PROCEDURE Field(nameAndType: S255));
001542      BEGIN
001543          SUPERSELF.Fields(Field);
001544          Field('layoutBox: TLayoutBox');
001545          Field('');
001546      END;
001547      {$S SgLayout}
001548      {$ENDC}
001549
001550
001551  {$S SgLayout}
001552      FUNCTION TLayPickSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
001553      BEGIN
001554          {$IFC fTrace}BP(10);{$ENDC}
001555          CASE cmdNumber OF
001556              uClear:
001557                  CanDoCommand := TRUE;
001558              OTHERWISE
001559                  CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
001560              END;
001561          {$IFC fTrace}EP;{$ENDC}
001562      END;
001563
001564
001565  {$S SgLayout}
001566      PROCEDURE TLayPickSelection.Deselect;
001567      BEGIN
001568          {$IFC fTrace}BP(11);{$ENDC}
001569          TPlannerView(SELF.view).currentLayoutBox := NIL;
001570          SELF.layoutBox.ChangeDragState(FALSE);
001571          SUPERSELF.Deselect;
001572          {$IFC fTrace}EP;{$ENDC}
001573      END;
001574
001575
001576  {$S SgLayout}
001577      PROCEDURE TLayPickSelection.Highlight(highTransit: THighTransit);
001578      BEGIN
001579          {$IFC fTrace}BP(11);{$ENDC}
```

```
001580          SELF.layoutBox.Highlight(highTransit);
001581          {$IFC fTrace}EP;{$ENDC}
001582      END;
001583
001584
001585  {$S SgLayout}
001586      PROCEDURE TLayPickSelection.MouseMove(mouseLPt: LPoint);
001587          VAR diffLPt:        LPoint;
001588              diffLRect:      LRect;
001589      BEGIN
001590         {$IFC fTrace}BP(11);{$ENDC}
001591             {How far did mouse move?}
001592              LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
001593             {Don't move past view boundaries}
001594              LRectMinusLRect(SELF.view.extentLRect, SELF.layoutBox.extentLRect, diffLRect);
001595              LRectHaveLPt(diffLRect, diffLPt);
001596
001597             {Move it if delta is nonzero}
001598              IF NOT EqualLPt(diffLPt, zeroLPt) THEN
001599                   BEGIN
001600            {$H-} OffsetLRect(SELF.boundLRect, diffLPt.h, diffLPt.v); {$H+}  {probably discard}
001601                   LPtPlusLPt(SELF.currLPt, diffLPt, mouseLPt);
001602                   SELF.currLPt := mouseLPt;
001603
001604                   SELF.layoutBox.Move(diffLPt);
001605                   END;
001606          {$IFC fTrace}EP;{$ENDC}
001607      END;
001608
001609
001610  {$S SgLayout}
001611      PROCEDURE TLayPickSelection.MouseRelease;
001612      VAR deltaLPt:       LPoint;
001613          noSelection:    TSelection;
001614          manipulee:      TImage;
001615          parent:         TDialogImage;
001616      BEGIN
001617          {$IFC fTrace}BP(11);{$ENDC}
001618          IF NOT EqualLPt(SELF.currLPt, SELF.anchorLPt) THEN
001619              BEGIN
001620              LPtMinusLPt(SELF.currLPt, SELF.anchorLPt, deltaLPt);
001621              SELF.window.PerformCommand(TLayMoveCmd.CREATE(NIL, SELF.Heap,
001622                              SELF.layoutBox, deltaLPt.h, deltaLPt.v));
001623              END;
001624
001625          manipulee := SELF.layoutBox.manipulee;
001626          IF manipulee <> NIL THEN
001627              manipulee.RecalcExtent;  {will send it up the line to its parents}
```

```
001628
001629          parent := SELF.layoutBox.parent;
001630          IF parent <> NIL THEN  {now send it up the line to my own parents}
001631              IF InClass(parent, TLayoutBox) THEN
001632                  parent.RecalcExtent;
001633
001634          IF NOT TPlannerView(SELF.layoutBox.view).retainPickedBox THEN
001635              SELF.Deselect;
001636          {$IFC fTrace}EP;{$ENDC}
001637      END;
001638
001639
001640  {$S SgLayout}
001641      PROCEDURE TLayPickSelection.Restore;
001642      BEGIN
001643          {$IFC fTrace}BP(11);{$ENDC}
001644          TPlannerView(SELF.view).currentLayoutBox := SELF.layoutBox;
001645          SUPERSELF.Restore;
001646          {$IFC fTrace}EP;{$ENDC}
001647      END;
001648
001649
001650  {$S DlgInit}
001651  END;
001652
001653
001654  METHODS OF TLayMoveCmd;
001655
001656  {$S SgLayout}
001657      FUNCTION  TLayMoveCmd.CREATE{(object: TObject; heap: THeap; itsLayoutBox: TLayoutBox;
001658                                    itsHOffset, itsVOffset: LONGINT): TLayMoveCmd};
001659          VAR retainPickedBox:    BOOLEAN;
001660              cmdPhase:           TCmdPhase;
001661      BEGIN
001662          {$IFC fTrace}BP(10);{$ENDC}
001663          IF object = NIL THEN
001664              object := NewObject(heap, THISCLASS);
001665          SELF := TLayMoveCmd(TCommand.CREATE(object, heap, uMoveLayoutBoxes, itsLayoutBox.view,
001666                      TRUE, revealSome));
001667
001668          WITH SELF DO
001669              BEGIN
001670              layoutBox := itsLayoutbox;
001671              hOffset := itsHOffset;
001672              vOffset := itsVOffset;
001673              retainPickedBox := TPlannerView(itsLayoutBox.view).retainPickedBox;
001674              WITH SELF DO
001675                  BEGIN
```

```
001676                  unHiliteBefore[doPhase] := FALSE;
001677                  hiliteAfter[doPhase] := FALSE;
001678                  END;
001679              END;
001680          {$IFC fTrace}EP;{$ENDC}
001681      END;
001682
001683
001684      {$IFC fDebugMethods}
001685  {$S DlgDbg}
001686      PROCEDURE TLayMoveCmd.Fields(PROCEDURE Field(nameAndType: S255));
001687      BEGIN
001688          SUPERSELF.Fields(Field);
001689          Field('layoutBox: TLayoutBox');
001690          Field('hOffset: LONGINT');
001691          Field('vOffset: LONGINT');
001692          Field('');
001693      END;
001694      {$ENDC}
001695
001696
001697  {$S SgLayout}
001698      PROCEDURE TLayMoveCmd.Perform(cmdPhase: TCmdPhase);
001699          VAR plannerView:    TPlannerView;
001700              panel:      TPanel;
001701              diffLPt:    LPoint;
001702      BEGIN
001703          {$IFC fTrace}BP(12);{$ENDC}
001704          IF cmdPhase <> doPhase THEN
001705              BEGIN
001706              WITH SELF DO {$H-}
001707                  CASE cmdPhase OF
001708                      redoPhase:
001709                          SetLPt(diffLPt, hOffset, vOffset);
001710                      undoPhase:
001711                          SetLPt(diffLPt, -hOffset, -vOffset);
001712                      END; {$H+}
001713              SELF.layoutBox.Move(diffLPt);
001714              SELF.layoutBox.manipulee.RecalcExtent;
001715              SELF.layoutBox.RecalcExtent;
001716              IF NOT TPlannerView(SELF.layoutBox.view).retainPickedBox THEN
001717                  SELF.layoutBox.view.panel.selection.Deselect;
001718              END;
001719          {$IFC fTrace}EP;{$ENDC}
001720      END;
001721
001722
001723  {$S DlgInit}
```

```
001724  END;
001725
001726
001727  METHODS OF TEditLegendSelection;
001728
001729
001730  {$S SgLayout}
001731      FUNCTION TEditLegendSelection.CREATE(object: TObject; heap: THeap; itsLegendLayoutBox:
001732                  TLegendLayoutBox; itsAnchorLPt: LPoint): TEditLegendSelection;
001733          VAR coSelection:        TSelection;
001734              paragraph:          TParagraph;
001735              paraImage:          TParaImage;
001736              hostLegend:         TLegend;
001737              textDialogImage:    TTextDialogImage;
001738              textImage:          TTextImage;
001739              textExtent:         LRect;
001740              typeStyle:          TTypeStyle;
001741              hostParagraph:      TParagraph;
001742
001743              PROCEDURE FindBiggestFont(VAR biggestTypeStyle: TTypeStyle);
001744              VAR styleChange:    TStyleChange;
001745  {$IFC libraryVersion <= 20}   {* * * P E P S I * * *}
001746                  fInfo:          TFInfo;
001747  {$ELSEC}                      {* *  S P R I N G  * *}
001748                  fInfo:          FontInfo;
001749  {$ENDC}
001750                  i:              INTEGER;
001751                  oldTallest:     INTEGER;
001752              BEGIN
001753                  oldTallest := 0;
001754                  FOR i := 1 TO hostParagraph.typeStyles.size - 1 DO
001755                      BEGIN
001756                      hostParagraph.typeStyles.GetAt(i, @styleChange);
001757                      hostParagraph.SetTypeStyle(styleChange.newStyle);
001758                      GetFontInfo(fInfo);
001759                      WITH fInfo DO
001760                          IF oldTallest < ascent + descent + leading THEN
001761                              BEGIN
001762                              oldTallest := ascent + descent + leading;
001763                              biggestTypeStyle := styleChange.newStyle;
001764                              END;
001765                      END;
001766              END;
001767
001768      BEGIN
001769          {$IFC fTrace}BP(11);{$ENDC}
001770          IF object = NIL THEN
001771              object := NewObject(heap, THISCLASS);
```

```
001772          SELF := TEditLegendSelection(TSelection.CREATE(object, heap, itsLegendLayoutBox.view,
001773                                                         layEditLegendSelectionKind, itsAnchorLPt));
001774          IF itsLegendLayoutBox.parent <> NIL THEN
001775              SELF.boundLRect := itsLegendLayoutBox.parent.extentLRect
001776          ELSE
001777              SELF.boundLRect := itsLegendLayoutBox.extentLRect;
001778
001779          SELF.legendLayoutBox := itsLegendLayoutBox;
001780          hostLegend := TLegend(itsLegendLayoutBox.manipulee);
001781          SELF.hostLegend := hostLegend;
001782          hostParagraph := hostLegend.paragraph;
001783
001784          SELF.suppressHost := itsLegendLayoutBox.suppressDrawingManipulee;
001785
001786          coSelection := SELF.panel.view.NoSelection;  {put non-NIL coSelection}
001787          SELF.coSelection := coSelection;
001788
001789          hostLegend.paragraph.StyleAt(0, typeStyle); {use paragraph's default if none else}
001790          FindBiggestFont(typeStyle);
001791
001792          SetParaExtent(hostLegend.paragraph, SELF.view, hostLegend.location, textExtent);
001793          textExtent.right := textExtent.left + 10 * SELF.view.res.h;   {ten inches wide}
001794          textDialogImage := TTextDialogImage.CREATE(NIL, heap, textExtent, noID, SELF.view,
001795              typeStyle, '');     {start off with an empty guy}
001796          SELF.textDialogImage := textDialogImage;
001797          textImage := textDialogImage.textImage;
001798          paraImage := TParaImage(textImage.imageList.First);
001799          paragraph := paraImage.paragraph;
001800
001801          paragraph.ReplPara(0,
001802              paragraph.size, hostLegend.paragraph, 0, hostLegend.paragraph.size);
001803
001804          paraImage.changed := TRUE;
001805          paraImage.InvalLinesWith(0, MAXINT);
001806
001807          textImage.RecomputeImages(actionNone, TRUE);
001808
001809          itsLegendLayoutBox.textDialogImage := textDialogImage;
001810          SELF.textDialogImage := textDialogImage;
001811          {$IFC fTrace}EP;{$ENDC}
001812      END;
001813
001814
001815  {$S SgLayout}
001816      FUNCTION  TEditLegendSelection.Clone(heap: THeap): TObject;
001817      BEGIN
001818          {$IFC fTrace}BP(11);{$ENDC}
001819          SELF.textDialogImage.ChangeRefCountBy(1);
```

```
001820          Clone := TEditLegendSelection(SUPERSELF.Clone(heap));
001821          {$IFC fTrace}EP;{$ENDC}
001822      END;
001823
001824
001825  {$S SgLayout}
001826      PROCEDURE TEditLegendSelection.Deselect;
001827      BEGIN
001828          {$IFC fTrace}BP(11);{$ENDC}
001829          SELF.legendLayoutBox.textDialogImage := NIL; {+SW+}
001830          SUPERSELF.Deselect;
001831          {$IFC fTrace}EP;{$ENDC}
001832      END;
001833
001834
001835  {$S SgLayout}
001836      PROCEDURE TEditLegendSelection.Free;
001837          VAR textDialogImage:    TTextDialogImage;
001838      BEGIN
001839          {$IFC fTrace}BP(11);{$ENDC}
001840          textDialogImage := SELF.textDialogImage;  {+SW+} {five lines out}
001841          SUPERSELF.Free;
001842          textDialogImage.ChangeRefCountBy(-1);
001843          {$IFC fTrace}EP;{$ENDC}
001844      END;
001845
001846
001847      {$IFC fDebugMethods}
001848      {$S DlgDbg}
001849      PROCEDURE TEditLegendSelection.Fields(PROCEDURE Field(nameAndType: S255));
001850      BEGIN
001851          SUPERSELF.Fields(Field);
001852          Field('legendLayoutBox: TLegendLayoutBox');
001853          Field('hostLegend: TLegend');
001854          Field('textDialogImage: TTextDialogImage');
001855          Field('suppressHost: BOOLEAN');
001856          Field('tripleClick: BOOLEAN');  {+SW+}
001857          Field('');
001858      END;
001859      {$ENDC}
001860
001861
001862  {$S SgLayout}
001863      FUNCTION TEditLegendSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
001864      BEGIN
001865          {$IFC fTrace}BP(10);{$ENDC}
001866          CASE cmdNumber OF
001867              uModern, uClassic,u20Pitch, u15Pitch, u12Pitch, u10Pitch, u12Point, u14Point, u18Point, u24Point,
```

```
001868              uPlain, uBold, uItalic, uUnderline, uShadow, uOutline:
001869                  IF SELF.hostLegend.usesSysFont THEN
001870                      CanDoCommand := FALSE
001871                  ELSE
001872                      CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
001873
001874              OTHERWISE
001875                  CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
001876              END;
001877          {$IFC fTrace}EP;{$ENDC}
001878      END;
001879
001880
001881  {$S SgLayout}
001882      PROCEDURE TEditLegendSelection.KeyBack(fWord: BOOLEAN);
001883      BEGIN
001884          {$IFC fTrace}BP(11);{$ENDC}
001885          SELF.coSelection.KeyBack(fWord);
001886          SELF.legendLayoutBox.RecalcExtent;  {will determine current width of the textDialogImage and
001887                                               adjust layout box + parents accordingly}
001888          {$IFC fTrace}EP;{$ENDC}
001889      END;
001890
001891
001892  {$S SgLayout}
001893      PROCEDURE TEditLegendSelection.KeyChar(ch: CHAR);
001894          VAR newExtent:      LRect;
001895              width:          INTEGER;
001896              paragraph:      TParagraph;
001897              i:              INTEGER;
001898      BEGIN
001899          {$IFC fTrace}BP(11);{$ENDC}
001900          paragraph := TParaImage(SELF.textDialogImage.textImage.imageList.First).paragraph;
001901          IF (paragraph.size < 255) OR
001902                  (NOT InClass(SELF.coSelection, TInsertionPoint)) THEN  {can accept more}
001903              BEGIN
001904              SELF.coSelection.KeyChar(ch);
001905
001906              SELF.legendLayoutBox.RecalcExtent;  {will determine current width of the textDialogImage and
001907                                                   adjust layout box + parents accordingly}
001908              END
001909          ELSE
001910              BEGIN
001911              process.ArgAlert(1, '255');
001912              process.Stop(phTooManyChars);
001913              END;
001914
001915          {$IFC fTrace}EP;{$ENDC}
```

```
001916      END;
001917
001918
001919  {$S SgLayout}
001920      PROCEDURE TEditLegendSelection.KeyEnter(dh, dv: INTEGER);
001921      BEGIN
001922          {$IFC fTrace}BP(11);{$ENDC}
001923          IF (dh <> 0) OR (dv <> 0) THEN
001924              SELF.KeyTab((dh < 0) OR (dv < 0));  {right and down keys are Forward}
001925          {$IFC fTrace}EP;{$ENDC}
001926      END;
001927
001928
001929  {$S SgLayout}
001930      PROCEDURE TEditLegendSelection.KeyReturn;
001931          VAR selection:  TSelection;
001932      BEGIN
001933          {$IFC fTrace}BP(11);{$ENDC}
001934          SELF.Deselect;
001935          {$IFC fTrace}EP;{$ENDC}
001936      END;
001937
001938
001939  {$S SgLayout}
001940      PROCEDURE TEditLegendSelection.MousePress(mouseLPT: LPoint); {+SW+}
001941          VAR selection: TSelection;
001942              textImage:  TTextImage;
001943      BEGIN
001944          {$IFC fTrace}BP(11);{$ENDC}
001945          IF clickState.clickCount < 3 THEN
001946              SUPERSELF.MousePress(mouseLPt)
001947          ELSE {triple click; force SelectAll}
001948              BEGIN
001949              SELF.tripleClick := TRUE;
001950              textImage := SELF.textDialogImage.textImage;
001951              SELF.Highlight(hOnToOff);
001952              SELF.coSelection.Become(
001953                      textImage.text.SelectAll(textImage));
001954              SELF.Highlight(hOffToOn);
001955              END;
001956          {$IFC fTrace}EP;{$ENDC}
001957      END;
001958
001959
001960  {$S SgLayout}
001961      PROCEDURE TEditLegendSelection.MouseMove(mouseLPT: LPoint); {+SW+}
001962      BEGIN
001963          {$IFC fTrace}BP(11);{$ENDC}
```

```
001964          IF NOT SELF.tripleClick THEN
001965              SUPERSELF.MouseMove(mouseLPt);
001966          {$IFC fTrace}EP;{$ENDC}
001967      END;
001968
001969
001970  {$S SgLayout}
001971      PROCEDURE TEditLegendSelection.MouseRelease; {+SW+}
001972      BEGIN
001973          {$IFC fTrace}BP(11);{$ENDC}
001974          IF SELF.tripleClick THEN
001975              SELF.tripleClick := FALSE
001976          ELSE
001977              SUPERSELF.MouseRelease;
001978          {$IFC fTrace}EP;{$ENDC}
001979      END;
001980
001981
001982  {$S SgLayout}
001983      PROCEDURE TEditLegendSelection.PerformCommand(command: TCommand; cmdPhase: TCmdPhase);
001984          VAR paragraph:  TParagraph;
001985      BEGIN
001986          {$IFC fTrace}BP(11);{$ENDC}
001987          SUPERSELF.PerformCommand(command, cmdPhase);
001988          IF SELF.hostLegend.usesSysFont THEN
001989              BEGIN
001990              paragraph := TParaImage(SELF.textDialogImage.textImage.imageList.First).paragraph;
001991              paragraph.NewStyle(0, paragraph.Size, sysTypeStyle);
001992              END;
001993          SELF.legendLayoutBox.RecalcExtent;  {will determine current width of the textDialogImage and
001994                                               adjust layout box + parents accordingly}
001995          {$IFC fTrace}EP;{$ENDC}
001996      END;
001997
001998
001999  {$S SgLayout}
002000      PROCEDURE TEditLegendSelection.Restore;
002001      BEGIN
002002          {$IFC fTrace}BP(11);{$ENDC}
002003          SELF.legendLayoutBox.textDialogImage := SELF.textDialogImage;
002004          SUPERSELF.Restore;
002005          {$IFC fTrace}EP;{$ENDC}
002006      END;
002007
002008
002009  {$S SgLayout}
002010      PROCEDURE TEditLegendSelection.Reveal(asMuchAsPossible: BOOLEAN);
002011          TYPE TXLRect = PACKED ARRAY [1..SIZEOF(LRect)] OF CHAR;
```

```
002012          VAR lr:         LRect;
002013              hMin:       INTEGER;
002014              vMin:       INTEGER;
002015      BEGIN
002016          {$IFC fTrace}BP(7);{$ENDC}
002017          SELF.coselection.boundLRect := SELF.boundLRect;
002018          SUPERSELF.Reveal(asMuchAsPossible);
002019          {$IFC fTrace}EP;{$ENDC}
002020      END;
002021
002022
002023  {$S DlgInit}
002024  END;
002025
002026
002027  METHODS OF TDialogDesignWindow;
002028
002029
002030  {$S DlgAlloc}
002031      FUNCTION TDialogDesignWindow.CREATE(object: TObject; heap: THeap;
002032                  itsHostDialogView: TDialogView): TDialogDesignWindow;
002033          VAR fromBox:    BOOLEAN;
002034              window:     TWindow;
002035              htLPt:      LPoint;
002036              height:     INTEGER;
002037              htPt:       Point;
002038      BEGIN
002039          {$IFC fTrace}BP(11);{$ENDC}
002040          window := itsHostDialogView.panel.window;
002041          fromBox := InClass(window, TDialogBox);
002042          IF fromBox THEN
002043              height :=  window.outerRect.bottom - window.outerRect.top + 15
002044          ELSE
002045              BEGIN
002046              WITH itsHostDialogView.extentLRect DO
002047        {$H-}     SetLPt(htLPt, 0, bottom - top);  {$H+}
002048              itsHostDialogView.screenPad.LPtToPt(htLPt, htPt);
002049              height := MIN(htPt.v + 15, screenBits.bounds.bottom - 30);
002050              END;
002051
002052          IF object = NIL THEN
002053              object := NewObject(heap, THISCLASS);
002054          SELF := TDialogDesignWindow(TDialogWindow.CREATE(object, heap, TRUE, height,
002055                                                           diAccept, diAccept, diRefuse));
002056
002057          WITH SELF DO
002058              BEGIN
002059              hostWindow := window;
```

```
002060              hostDialogView := itsHostDialogView;
002061              fromDialogBox := fromBox;
002062              END;
002063          {$IFC fTrace}EP;{$ENDC}
002064      END;
002065
002066
002067      {$IFC fDebugMethods}
002068      {$S DlgDbg}
002069      PROCEDURE TDialogDesignWindow.Fields(PROCEDURE Field(nameAndType: S255));
002070      BEGIN
002071          SUPERSELF.Fields(Field);
002072          Field('hostWindow: TWindow');
002073          Field('hostDialogView: TDialogView');
002074          Field('fromDialogBox: BOOLEAN');
002075          Field('');
002076      END;
002077     {$ENDC}
002078
002079
002080  {$S DlgLayout}
002081      FUNCTION TDialogDesignWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
002082      BEGIN
002083          {$IFC fTrace}BP(11);{$ENDC}
002084          CASE cmdNumber OF
002085              uEditDialog:
002086                  CanDoCommand := FALSE;  {override SUPERSELF}
002087              uStopEditDialog:
002088                  CanDoCommand := TRUE;
002089              OTHERWISE
002090                  CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
002091              END;
002092          {$IFC fTrace}EP;{$ENDC}
002093       END;
002094
002095
002096  {$S DlgLayout}
002097      FUNCTION  TDialogDesignWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;
002098      BEGIN
002099          {$IFC fTrace}BP(12);{$ENDC}
002100          CASE cmdNumber OF
002101              uStopEditDialog:
002102                  BEGIN
002103                  SELF.RelinquishControl;
002104                  NewCommand := NIL;
002105                  END;
002106              OTHERWISE
002107                  NewCommand := SUPERSELF.NewCommand(cmdNumber);
```

```
002108              END;
002109          {$IFC fTrace}EP;{$ENDC}
002110      END;
002111
002112
002113  {$S DlgLayout}
002114      PROCEDURE TDialogDesignWindow.RelinquishControl;
002115                  {not yet:  install in resourceFile}
002116          VAR panel:              TPanel;
002117              plannerView:        TPlannerView;
002118              dialogWindow:       TDialogWindow;
002119              newHeight:          INTEGER;
002120              noSelection:        TSelection;
002121              newBotRight:        point;
002122       BEGIN
002123          {$IFC fTrace}BP(11);{$ENDC}
002124          panel := SELF.selectPanel;
002125          panel.selection.Deselect;      {should incorporate last text edit, if any}
002126          panel.window.CommitLast;
002127          panel.BeginSelection;          {previous didn't really quite do it yet}
002128
002129          currentWindow.TakeDownDialogBox;                              {take down layout dialog}
002130
002131          IF SELF.fromDialogBox THEN {editing a dialog box--copy the resizing back to the dialog Window}
002132              BEGIN
002133              plannerView := TPlannerView(panel.currentView);
002134              newHeight := panel.innerRect.bottom;
002135              dialogWindow := TDialogWindow(plannerView.viewBeingPlanned.panel.window);
002136              SetPt(newBotRight, screenBits.bounds.right, newHeight);       {transfer its current...}
002137              currentWindow.PutUpDialogBox(dialogWindow);
002138              dialogWindow.ResizeTo(newBotRight);                           {height to main dialog}
002139              IF dialogWindow.selectPanel.selection.kind <> nothingKind THEN
002140                  currentWindow.selectWindow := dialogWindow; {=}
002141              END;
002142
002143          SELF.selectPanel.view := NIL;     {kludge to avoid clobbering the main view}
002144          currentWindow.Focus; {necessary to avoid a later popFocus trying to focus our now departed
002145                                DialogDesignWindow}
002146          SELF.Free; {!}
002147          {$IFC fTrace}EP;{$ENDC}
002148       END;
002149
002150
002151  {$S DlgLayout}
002152      PROCEDURE TDialogDesignWindow.Resize(moving: BOOLEAN);
002153          VAR view:           TView;
002154              extentLRect:    LRect;
002155      BEGIN
```

```
002156          {$IFC fTrace}BP(11);{$ENDC}
002157          SUPERSELF.Resize(moving); {moving is always FALSE of course}
002158          IF SELF.hostWindow.isResizable THEN
002159              {do nothing, I guess}
002160          ELSE
002161              BEGIN
002162              view := SELF.selectPanel.currentView;
002163              extentLRect := view.extentLRect;
002164              extentLRect.bottom := SELF.selectPanel.innerRect.bottom;
002165              view.Resize(extentLRect);     {that'll be the layout view}
002166              SELF.hostDialogView.Resize(extentLRect);
002167              END;
002168          {$IFC fTrace}EP;{$ENDC}
002169      END;
002170
002171
002172  {$S DlgLayout}
002173      PROCEDURE TDialogDesignWindow.SeizeControl;
002174          VAR dialogView:     TDialogView;
002175              panel:          TPanel;
002176              children:       TList;
002177              imageList:      TList;
002178              dialogBox:      TDialogBox;
002179              savedSetting:   BOOLEAN;
002180      BEGIN
002181          {$IFC fTrace}BP(11);{$ENDC}
002182          dialogView := SELF.hostDialogView;
002183          children := dialogView.rootDialog.children;
002184          IF SELF.fromDialogBox THEN
002185              BEGIN
002186              dialogBox := currentWindow.dialogBox;
002187              savedSetting := dialogBox.freeOnDismissal;
002188              dialogBox.freeOnDismissal := FALSE;
002189              currentWindow.TakeDownDialogBox;
002190              dialogBox.freeOnDismissal := savedSetting;
002191              END;
002192
002193
002194          panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, SELF.innerRect.right - 23, [aScroll, aBar],
002195                      [aScroll, aBar]);
002196          SELF.controlPanel := panel;
002197
002198          CASE children.Size OF
002199              0:          ABCBreak('SeizeControl, empty children of dialog view', 0);
002200              1:          imageList := TDialog(children.First).children;
002201              OTHERWISE   imageList := children;
002202              END; {case}
002203
```

```
002204          TPlannerView.CREATE(NIL, SELF.Heap, dialogView, panel, FALSE, FALSE).Init(imageList);
002205
002206          currentWindow.PutUpDialogBox(SELF);
002207          panel.BeginSelection;
002208          {$IFC fTrace}EP;{$ENDC}
002209      END;
002210
002211
002212  {$S DlgInit}
002213  END;
002214

End of File -- Lines: 2214 Characters: 68581
```

```
================================================================================================
FILE: "LIBTK/UDIALOG4.TEXT"
================================================================================================

{UDialog4}   {Handles Page Headings & Page Margins}  {Copyright 1984 by Apple Computer, INC}

(*

 ORDER OF METHODS:

      CLASS              SUBCLASS OF      WHICH IS IN
    -----------------   -------------    -----------
    TStdPrintManager     TPrintManager    UABC3
    TLegendHeading       THeading         UABC3
    TPageDesignWindow    TDialogWindow    UDialog2
    TPagePlannerView     TPlannerView     UDialog3
    TPageLayoutBox       TLayoutBox       UDialog3
    TLgHdngLayoutBox     TPageLayoutBox   UDialog4
    TPageStatusDialog    TDialog          UDialog2

*)
{04/25/84 19:00 TLgHdngLayoutBox.MousePress -- don't invalidate the layout box
                   TPageStatusDialog.CREATE explicitly sets extentLRect's topLeft to zeroLPt}
{04/24/84 23:51  TPageDesignWindow.CREATE sets the status view's scrollPastEnd to zeroPt}
{04/24/84 18:00  TPageDesignWindow.CREATE allows scrolling in the status panel}
{04/23/84 12:18  In TStdPrintManager.SetDfltHeadings, supply a blank space before and after the
                   the '{TITLE)', so that a select-all followed by a font change will result in
                   the new font applying to the substituted string at print or page-preview time}
{04/17/84 17:16  Make the TPagePlannerView show no gray at the end.}
{04/15/84 0200   TPageDesignWindow.NewCommand frees panel's undoSelection as well as main selection}
{04/14/84 03:00  Removed pilotHeading from TPageLayoutBox; removed TPageLayoutBox.Free, consequently,
                         as well as TPageLayoutBox.Fields
                    Offset master as well as current Legend in TLegendHeading.OffsetBy}
{changed 04/14/84 0102 Major rewrite of TLgHdngLayoutBox.RecalcExtent & TP }
{changed 04/13/84 2230 TLgHdngLayoutBox.RecalcExtent doesn't call SetParaExtent; just uses legend's extent
                        & TPageLayoutBox.FreeManipulee sets SELF.manipulee to NIL after freeing
                        & TPageDesignWindow.NewCommand sets plannerView.currentLayoutBox to NIL
                            after freeing the LgHdngLayoutBox...}
{changed 04/13/84 1630 Changed to not using bolding on the margins-dialog heading}
{changed 04/11/84 2315 Do TopToBaseLine stuff in hdngs recalculation only if not fExperimenting...}
{changed 04/11/84 1700 Use dfltNewHeading global var in laucnhing new heading in TDialogDesignWindow.NewCmd,
                          and varPage and varTitle in TStdPrintManager.SetDfltHeadings;
                       In TPageStatusDialog.CREATE, use a CONST depending on libraryVersion to determine
                          the spacing between the boxes in the margins checkbox dialog}
{04/04/84 2300  Spring Prelim Release}
{01/29/84 1800  RELEASE TK8D}
{12/21/83 1657  RELEASE TK8A}
```

```
000044
000045
000046  METHODS OF TStdPrintManager;
000047
000048
000049  {$S DlgAlloc}
000050      FUNCTION TStdPrintManager.CREATE(object: TObject; heap: THeap): TStdPrintManager;
000051      BEGIN
000052          {$IFC fTrace}BP(11);{$ENDC}
000053          IF object = NIL THEN
000054              object := NewObject(heap, THISCLASS);
000055          SELF := TStdPrintManager(TPrintManager.CREATE(object, heap));
000056          {$IFC fTrace}EP;{$ENDC}
000057      END;
000058
000059
000060  {$S DlgAlloc}
000061      PROCEDURE  TStdPrintManager.SetDfltHeadings;
000062          CONST   topFudge = 0;
000063                  bottomFudge = 0;
000064          VAR anOffset:   LPoint;
000065              margins:    LRect;
000066      BEGIN
000067          {$IFC fTrace}BP(7);{$ENDC}
000068          margins := SELF.pageMargins;
000069          SetLPt(anOffset, 0, (margins.top + topFudge) DIV 2);
000070          SELF.headings.InsLast(TLegendHeading.CREATE(NIL, SELF.Heap, SELF, CONCAT(' {', varTitle, '} '), {+SW+}
000071              stdHdngTypeStyle, aTopCenter, anOffset, stdHdngBorders));
000072
000073          SetLPt(anOffset, 0, - (ABS(margins.bottom + bottomFudge) DIV 2));
000074          SELF.headings.InsLast(TLegendHeading.CREATE(NIL, SELF.Heap, SELF,  CONCAT('-{', varPage, '}-'),
000075              stdHdngTypeStyle, aBottomCenter, anOffset, stdHdngBorders));
000076
000077          {$IFC fTrace}EP;{$ENDC}
000078      END;
000079
000080
000081  {$S DlgAlloc}
000082      PROCEDURE TStdPrintManager.Init(itsMainView: TView; itsDfltMargins: LRect);
000083      BEGIN
000084          {$IFC fTrace}BP(7);{$ENDC}
000085          SUPERSELF.Init(itsMainView, itsDfltMargins);
000086          SELF.canEditPages := TRUE;
000087          {$IFC fTrace}EP;{$ENDC}
000088      END;
000089
000090
000091  {$S HdgMarg}
```

```
000092      PROCEDURE  TStdPrintManager.EnterPageEditting;
000093          VAR window:             TWindow;
000094              pageDesignWindow:   TPageDesignWindow;
000095              pagePlannerView:    TPagePlannerView;
000096      BEGIN
000097          {$IFC fTrace}BP(7);{$ENDC}
000098          window := SELF.view.panel.window;
000099          window.CommitLast;
000100          IF SELF.layoutDialogBox = NIL THEN
000101              BEGIN
000102              pageDesignWindow := TPageDesignWindow.CREATE(NIL, SELF.Heap, SELF.view);
000103              SELF.layoutDialogBox := pageDesignWindow;
000104              END;
000105          window.PutUpDialogBox(SELF.layoutDialogBox);
000106          {$IFC fTrace}EP;{$ENDC}
000107      END;
000108
000109
000110  {$S TK2Start}
000111      PROCEDURE  TStdPrintManager.ReactToPrinterChange;
000112      BEGIN
000113          {$IFC fTrace}BP(7);{$ENDC}
000114          SUPERSELF.ReactToPrinterChange;
000115          IF SELF.layoutDialogBox <> NIL THEN
000116              TPageDesignWindow(SELF.layoutDialogBox).layoutPanel.view.Resize(SELF.pageView.extentLRect);
000117          {$IFC fTrace}EP;{$ENDC}
000118      END;
000119
000120
000121  {$S DlgInit}
000122  END;
000123
000124
000125  METHODS OF TLegendHeading;
000126
000127  {$S DlgAlloc}
000128      FUNCTION  TLegendHeading.CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
000129                      itsString: S255; itsTypeStyle: TTypeStyle;
000130                      itsPageAlignment: TPageAlignment; itsOffsetFromAlignment: LPoint;
000131                      itsBorders: Rect): TLegendHeading;
000132          VAR newMaster:  TLegend;
000133              newCurrent: TLegend;
000134              extent:     LRect;
000135              view:       TView;
000136      BEGIN
000137          {$IFC fTrace}BP(7);{$ENDC}
000138          view := itsPrintManager.pageView;
000139          SetLRect(extent, 0, 0, 100, 100);  {meaningless at this point}
```

```
000140
000141          IF object = NIL THEN
000142              object := NewObject(heap, THISCLASS);
000143          SELF := TLegendHeading(THeading.CREATE(object, heap, itsPrintManager, extent, itsPageAlignment,
000144                                                 itsOffsetFromAlignment));
000145
000146          newMaster:= TLegend.CREATE(NIL, heap, itsString, view, zeroLPt, itsTypeStyle);
000147          newCurrent := TLegend.CREATE(NIL, heap, itsString, view,  zeroLPt, itsTypeStyle);
000148          newMaster.HaveView(view);
000149          newCurrent.HaveView(view);
000150
000151          SetParaExtent(newMaster.paragraph, view, zeroLPt, extent);
000152
000153          WITH SELF DO
000154              BEGIN
000155              masterLegend := newMaster;
000156              currentLegend := newCurrent;
000157              borders := itsBorders;
000158              minPage := 1;   {readjusts from std 2, for demo purposes}
000159              topToBaseline :=  - itsBorders.top - extent.top; {both tops are negative}
000160              END;
000161
000162          newMaster.wouldBeDraggable := FALSE;
000163          newCurrent.wouldBeDraggable := FALSE;
000164
000165          SELF.RecalcExtent;
000166          {$IFC fTrace}EP;{$ENDC}
000167      END;
000168
000169
000170  {$S HdgMarg}
000171      PROCEDURE TLegendHeading.Free;
000172      BEGIN
000173          {$IFC fTrace}BP(7);{$ENDC}
000174          Free(SELF.masterLegend);
000175          Free(SELF.currentLegend);
000176          SUPERSELF.Free;
000177          {$IFC fTrace}EP;{$ENDC}
000178      END;
000179
000180
000181      {$IFC fDebugMethods}
000182      {$S DlgDbg}
000183      PROCEDURE TLegendHeading.Fields(PROCEDURE Field(nameAndType: S255));
000184      BEGIN
000185          SUPERSELF.Fields(Field);
000186          Field('masterLegend: TLegend');
000187          Field('currentLegend: TLegend');
```

```
000188          Field('topToBaseline: INTEGER');
000189          Field('borders: Rect');
000190          Field('');
000191      END;
000192      {$ENDC}
000193
000194
000195  {$S DlgRes}
000196      PROCEDURE TLegendHeading.AdjustForPage(pageNumber: LONGINT; editing: BOOLEAN);
000197          VAR currS255:           S255;
000198              aVariable:          S255;
000199              leftBracePos:       INTEGER;
000200              rightBracePos:      INTEGER;
000201              newValue:           S255;
000202              restOfString:       S255;
000203              newExtent:          LRect;
000204              currentParagraph:   TParagraph;
000205              masterParagraph:    TParagraph;
000206              substituted:        BOOLEAN;
000207              lastPosition:       INTEGER;
000208      BEGIN
000209          {$IFC fTrace}BP(9);{$ENDC}
000210          substituted := FALSE;  {still flawed}
000211          lastPosition := 0;
000212          SELF.masterLegend.GetString(currS255);
000213          currentParagraph := SELF.currentLegend.paragraph;
000214          masterParagraph := SELF.masterLegend.paragraph;
000215          currentParagraph.ReplPara(0, currentParagraph.size, masterParagraph, 0,
000216              masterParagraph.size);  {download entire master into current}
000217          IF NOT editing THEN
000218              BEGIN
000219              REPEAT
000220              leftBracePos := POS('{', currS255);
000221              IF leftBracePos > 0 THEN
000222                  IF leftBracePos < lastPosition THEN {was within the previous variable}
000223                      currS255[leftBracePos] := '$'    {... so we won't get it next time}
000224                  ELSE
000225                      BEGIN
000226                      restOfString := COPY(currS255, leftBracePos + 1, LENGTH(currS255) - leftBracePos);
000227                      rightBracePos := POS('}', restOfString);
000228                      IF rightBracePos > 0 THEN
000229                          BEGIN
000230                          aVariable := COPY(restOfString, 1, rightBracePos - 1);
000231                          SELF.printManager.view.SetFunctionValue(aVariable, newValue);
000232                          substituted := TRUE;
000233                          currentParagraph.ReplPString(leftBracePos - 1, rightBracePos + 1,
000234                              @newValue);
000235
```

```
000236                          DELETE(currS255, leftBracePos, rightBracePos + 1);  {get rid of the var code}
000237                          INSERT(newValue, currS255, leftBracePos);  {substitute the variable's value}
000238                          currS255[leftBracePos] := '$';
000239                          lastPosition := leftBracePos + LENGTH(newValue);
000240                          END
000241                      ELSE
000242                          lastPosition := LENGTH(currS255) + 1;
000243                      END;
000244              UNTIL
000245                  leftBracePos = 0;
000246
000247              END   {not editing}
000248
000249          ELSE  {editing}
000250              SELF.masterLegend.GetBoxRight;
000251
000252          SELF.RecalcExtent;  {tells currentLegend to get box right, then adds in my borders}
000253         {we only need worry about our extentLRect, our location, and our current legend all
000254           being in synch; THeading.LocateOnPage will then find the exact page location,
000255           taking into account my offsetFromAlignment}
000256
000257          {$IFC fTrace}EP;{$ENDC}
000258      END;
000259
000260
000261  {$S DlgRes}
000262      PROCEDURE TLegendHeading.Draw;
000263      BEGIN
000264          {$IFC fTrace}BP(9);{$ENDC}
000265          IF SELF.ShouldFrame THEN
000266              FrameLRect(SELF.extentLRect);
000267          SELF.currentLegend.Draw;
000268          {$IFC fTrace}EP;{$ENDC}
000269      END;
000270
000271
000272  {$S HdgMarg}
000273      FUNCTION TLegendHeading.LaunchLayoutBox(view: TView): TImage;
000274      BEGIN
000275          {$IFC fTrace}BP(10);{$ENDC}
000276          LaunchLayoutBox := TLgHdngLayoutBox.CREATE(NIL, SELF.Heap, view, SELF);
000277          {$IFC fTrace}EP;{$ENDC}
000278      END;
000279
000280
000281  {$S HdgMarg}
000282      PROCEDURE TLegendHeading.OffsetBy(deltaLPt: LPoint);
000283      BEGIN
```

```
000284          {$IFC fTrace}BP(9);{$ENDC}
000285          SELF.currentLegend.OffsetBy(deltaLPt);
000286          SUPERSELF.OffsetBy(deltaLPt);
000287          {$IFC fTrace}EP;{$ENDC}
000288      END;
000289
000290
000291  {$S TK2Start}
000292      PROCEDURE TLegendHeading.RecalcExtent;
000293          VAR newExtent:  LRect;
000294      BEGIN
000295          {$IFC fTrace}BP(9);{$ENDC}
000296          SELF.currentLegend.GetBoxRight;
000297          LRectAddBorders(SELF.currentLegend.extentLRect, SELF.borders, newExtent);
000298          SELF.Resize(newExtent);
000299          {$IFC fTrace}EP;{$ENDC}
000300      END;
000301
000302
000303  {$S DlgRes}
000304      FUNCTION TLegendHeading.ShouldFrame;
000305      BEGIN
000306          {$IFC fTrace}BP(9);{$ENDC}
000307          ShouldFrame := FALSE;
000308          {$IFC fTrace}EP;{$ENDC}
000309      END;
000310
000311
000312  {$S DlgInit}
000313  END;
000314
000315
000316  METHODS OF TPageDesignWindow;
000317
000318
000319  {$S DlgAlloc}
000320      FUNCTION TPageDesignWindow.CREATE(object: TObject; heap: THeap; itsHostView: TView): TPageDesignWindow;
000321          CONST   cPgWindowHeight = 340;
000322                  cPgControlHeight = 130; {height of the control (status) panel}
000323                  cHtStatusView = 220;
000324
000325          VAR controlPanel:   TPanel;
000326              layoutPanel:    TPanel;
000327              hdngDialog:     THeadingDialog;
000328              plannerView:    TPlannerView;
000329              dialogView:     TDialogView;
000330              extentLRect:    LRect;
000331      BEGIN
```

```
000332          {$IFC fTrace}BP(11);{$ENDC}
000333          IF object = NIL THEN
000334              object := NewObject(heap, THISCLASS);
000335          SELF := TPageDesignWindow(TDialogWindow.CREATE(object, heap, TRUE, cPgWindowHeight, diAccept,
000336                                                         diAccept, diRefuse));
000337
000338          SELF.hostView := itsHostView;
000339
000340          layoutPanel := TPanel.CREATE(NIL, heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
000341          plannerView := TPagePlannerView.CREATE(NIL, heap, itsHostView.printManager, layoutPanel);
000342          SELF.layoutPanel := layoutPanel;
000343
000344          controlPanel  := layoutPanel.Divide(v, cPgControlHeight, pixelsFromEdge,
000345                  [userCanResizeIt], 10 {min size}, [aScroll], [aScroll]);  {+SW+}
000346          SELF.controlPanel := controlPanel;
000347          SetLRect(extentLRect, 0, 0, screenBits.bounds.right, cHtStatusView);
000348          dialogView := TDialogView.CREATE(NIL, heap, extentLRect, controlPanel, NIL, screenRes);
000349          dialogView.scrollPastEnd := zeroPt; {+SW+}
000350          dialogView.AddDialog(TPageStatusDialog.CREATE(NIL, heap, dialogView.panel));
000351
000352          {$IFC fTrace}EP;{$ENDC}
000353      END;
000354
000355
000356      {$IFC fDebugMethods}
000357      PROCEDURE TPageDesignWindow.Fields(PROCEDURE Field(nameAndType: S255));
000358      BEGIN
000359          SUPERSELF.Fields(Field);
000360          Field('hostView: TView');
000361          Field('layoutPanel: TPanel');
000362          Field('');
000363      END;
000364     {$ENDC}
000365
000366
000367      PROCEDURE TPageDesignWindow.Disappear;
000368          VAR panel:  TPanel;
000369       BEGIN
000370          {$IFC fTrace}BP(11);{$ENDC}
000371          panel := TPagePlannerView(SELF.layoutPanel.view).viewBeingPlanned.panel;
000372          IF panel.previewMode = mPrvwMargins THEN {make sure headings are updated}
000373              panel.Invalidate;
000374          SUPERSELF.Disappear;
000375          {$IFC fTrace}EP;{$ENDC}
000376       END;
000377
000378
000379  {$S HdgMarg}
```

```
000380      FUNCTION  TPageDesignWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;
000381      {unusually, uClear is armed by TLayPickSelection.NewCommand but dealt with by the PageDesignWindow}
000382          VAR s:              TListScanner;
000383              layoutBox:      TLayoutBox;
000384              plannerView:    TPlannerView;
000385              noSelection:    TSelection;
000386              command:        TCommand;
000387              selectedBox:    TLayoutBox;
000388              panel:          TPanel;
000389      BEGIN
000390          {$IFC fTrace}BP(11);{$ENDC}
000391          CASE cmdNumber OF
000392              uClear:  {not undoable at present...}
000393                  BEGIN
000394                  SELF.CommitLast; {The committal might require a to-be-freed textImage}
000395                  plannerView := TPlannerView(SELF.layoutPanel.view);
000396                  panel := plannerView.panel;
000397                  selectedBox := plannerView.currentLayoutBox;
000398                  s := plannerView.rootDialogImage.children.Scanner;
000399                  WHILE s.Scan(layoutBox) DO
000400                      IF layoutBox = selectedBox THEN
000401                          BEGIN
000402                          panel.selection.Deselect;
000403                          noSelection := panel.undoSelection.FreedAndReplacedBy(panel.view.NoSelection);
000404                          panel.InvalLRect(layoutBox.extentLRect);
000405                          layoutBox.FreeManipulee;   {Delete heading from the printManager}
000406                          s.Delete(TRUE);            {Delete heading's layout box from the plannerView}
000407                          s.Done;
000408                          END;
000409                  command := TCommand.CREATE(NIL, plannerView.Heap, uClear, plannerView, FALSE,
000410                                             revealNone);
000411                  NewCommand := command;
000412                  plannerView.currentLayoutBox := NIL;
000413                  TPageStatusDialog(SELF.mainDialog).currentHeading := NIL;
000414                  END;
000415
000416              OTHERWISE
000417                  NewCommand := SUPERSELF.NewCommand(cmdNumber);
000418              END;
000419          {$IFC fTrace}EP;{$ENDC}
000420      END;
000421
000422  {$S DlgInit}
000423  END;
000424
000425
000426  METHODS OF TPagePlannerView;
000427
```

```
000428
000429  {$S DlgAlloc}
000430      FUNCTION TPagePlannerView.CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
000431                                       itsPanel: TPanel): TPagePlannerView;
000432      BEGIN
000433          {$IFC fTrace}BP(11);{$ENDC}
000434          IF object = NIL THEN
000435              object := NewObject(heap, THISCLASS);
000436          SELF := TPagePlannerView(TPlannerView.CREATE(object, heap, itsPrintManager.pageView, itsPanel,
000437                                                       FALSE, TRUE));
000438          PushFocus;
000439          TPane(itsPrintManager.view.panel.panes.First).Focus;  {so that thePad will be set to something}
000440          SELF.Init(itsPrintManager.headings);
000441          PopFocus;
000442
000443          SELF.scrollPastEnd := zeroPt;
000444          {$IFC fTrace}EP;{$ENDC}
000445      END;
000446
000447
000448  {$S HdgMarg}
000449      PROCEDURE TPagePlannerView.Draw;
000450          VAR contentLRect:   LRect;
000451              pat:            Pattern;
000452              contentRect:    Rect;
000453      BEGIN
000454          {$IFC fTrace}BP(11);{$ENDC}
000455          contentLRect := SELF.viewBeingPlanned.printManager.contentLRect;   {screen embellishments}
000456          thePad.LPatToPat(marginPattern, pat);
000457          thePad.LRectToRect(contentLRect, contentRect);
000458          FillRect(contentRect, pat);
000459
000460          PenNormal;
000461          FrameLRect(SELF.extentLRect);
000462          FrameLRect(contentLRect);
000463
000464          SUPERSELF.Draw;   {draw LayoutBoxes}
000465          {$IFC fTrace}EP;{$ENDC}
000466      END;
000467
000468
000469  {$S DlgInit}
000470  END;
000471
000472
000473  METHODS OF TPageLayoutBox;
000474
000475  {$S HdgMarg}
```

```
000476      FUNCTION  TPageLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView; itsHeading: THeading;
000477                  itsResizable: BOOLEAN): TPageLayoutBox;
000478              VAR baseExtent: LRect;
000479      BEGIN
000480          {$IFC fTrace}BP(7);{$ENDC}
000481          baseExtent := itsHeading.extentLRect;
000482          baseExtent.top := baseExtent.top + stdSlimTitleHeight;
000483          IF object = NIL THEN
000484              object := NewObject(itsHeading.Heap, THISCLASS);
000485          SELF := TPageLayoutBox(TLayoutBox.CREATE(object, heap, baseExtent, noID, NIL,
000486                                                   itsView, itsHeading, stdPlainBorders, itsResizable,
000487                                                   TRUE, TRUE));
000488          {$IFC fTrace}EP;{$ENDC}
000489      END;
000490
000491
000492      {$IFC fDebugMethods}
000493      {$S DlgDbg}
000494      PROCEDURE TPageLayoutBox.Fields(PROCEDURE Field(nameAndType: S255));
000495      BEGIN
000496          SUPERSELF.Fields(Field);
000497          Field('');
000498      END;
000499      {$ENDC}
000500
000501
000502  {$S HdgMarg}
000503      PROCEDURE TPageLayoutBox.FreeManipulee;
000504          VAR s:          TListScanner;
000505              heading:    THeading;
000506      BEGIN
000507          {$IFC fTrace}BP(10);{$ENDC}
000508          s := TPlannerView(SELF.view).viewBeingPlanned.view.printManager.headings.Scanner;
000509          WHILE s.Scan(heading) DO
000510              IF heading = SELF.manipulee THEN
000511                  BEGIN
000512                  s.Delete(TRUE);
000513                  s.Done;
000514                  SELF.manipulee := NIL;
000515                  END;
000516          {$IFC fTrace}EP;{$ENDC}
000517      END;
000518
000519
000520  {$S HdgMarg}
000521      PROCEDURE TPageLayoutBox.TabGrabbed;
000522          VAR heading:            THeading;
000523              pageStatusDialog:   TPageStatusDialog;
```

```
000524      BEGIN
000525          {$IFC fTrace}BP(10);{$ENDC}
000526          heading := THeading(SELF.manipulee);
000527          pageStatusDialog := TPageStatusDialog(TDialogView(TDialogWindow(SELF.view.panel.window
000528              ).controlPanel.view).rootDialog.children.First);
000529          IF heading <> pageStatusDialog.currentHeading THEN
000530              BEGIN
000531              WITH heading DO
000532             {$H-}    pageStatusDialog.SetHeadingParms(oddOnly, evenOnly, pageAlignment, minPage, maxPage); {$H+}
000533              pageStatusDialog.currentHeading := heading;
000534              END;
000535          {$IFC fTrace}EP;{$ENDC}
000536      END;
000537
000538
000539  {$S DlgInit}
000540  END;
000541
000542
000543  METHODS OF TLgHdngLayoutBox;
000544
000545  {$S HdgMarg}
000546      FUNCTION  TLgHdngLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView;
000547                  itsLegendHeading: TLegendHeading): TLgHdngLayoutBox;
000548          VAR myExtent:           LRect;
000549              itsTitleTab:        TTitleTab;
000550              masterLegend:       TLegend;
000551              legendLayoutBox:    TLegendLayoutBox; {= SELF.children.First}
000552      BEGIN
000553          {$IFC fTrace}BP(7);{$ENDC}
000554          itsLegendHeading.AdjustForPage(0, TRUE);
000555          itsLegendHeading.LocateOnPage(TRUE);
000556          masterLegend := itsLegendHeading.masterLegend;
000557          masterLegend.location := itsLegendHeading.currentLegend.location;
000558          masterLegend.GetBoxRight;
000559
000560          LRectAddBorders(masterLegend.extentLRect, itsLegendHeading.borders, myExtent);
000561
000562          IF object = NIL THEN
000563              object := NewObject(heap, THISCLASS);
000564          SELF := TLgHdngLayoutBox(TImageWithID.CREATE(object, heap, myExtent, noID,
000565                  itsView, TRUE));
000566
000567          itsTitleTab := TTitleTab.CREATE(NIL, heap, SELF,  stdSlimTitleHeight, noID);
000568
000569          WITH SELF DO
000570              BEGIN
000571              titleTab := itsTitleTab;
```

```
000572              manipulee := itsLegendHeading;
000573              suppressDrawingManipulee := TRUE;
000574              wouldMakeSelection := TRUE;
000575              isResizable := FALSE;
000576              isDraggable := TRUE;
000577              shouldFrame := TRUE;
000578              borders := zeroRect;
000579              END;
000580
000581          legendLayoutBox := TLegendLayoutBox(itsLegendHeading.masterLegend.LaunchLayoutBox(itsView));
000582          SELF.legendLayoutBox := legendLayoutBox;
000583          SELF.AddImage(legendLayoutBox);
000584
000585          {$IFC fTrace}EP;{$ENDC}
000586      END;
000587
000588
000589      {$IFC fDebugMethods}
000590      {$S DlgDbg}
000591      PROCEDURE TLgHdngLayoutBox.Fields(PROCEDURE Field(nameAndType: S255));
000592      BEGIN
000593          SUPERSELF.Fields(Field);
000594          Field('legendLayoutBox: TLegendLayoutBox');
000595          Field('');
000596      END;
000597      {$ENDC}
000598
000599
000600  {$S HdgMarg}
000601      FUNCTION TLgHdngLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber;
000602          BEGIN
000603              {$IFC fTrace}BP(11);{$ENDC}
000604              IF SELF.Hit(mouseLPt) THEN
000605                  IF SELF.titleTab.Hit(mouseLPt) THEN
000606                      CursorAt := arrowCursor
000607                  ELSE
000608                      CursorAt := textCursor
000609              ELSE
000610                  CursorAt := noCursor;
000611              {$IFC fTrace}EP;{$ENDC}
000612          END;
000613
000614
000615  {$S HdgMarg}
000616      PROCEDURE TLgHdngLayoutBox.Draw;
000617      BEGIN
000618          {$IFC fTrace}BP(11);{$ENDC}
000619          IF LRectIsVisible(SELF.extentLRect) THEN
```

```
000620              BEGIN
000621              SELF.titleTab.Draw;
000622              PenNormal;
000623              FrameLRect(SELF.extentLRect);  {draw overall box}
000624              SELF.legendLayoutBox.Draw;
000625              END;
000626          {$IFC fTrace}EP;{$ENDC}
000627      END;
000628
000629
000630  {$S HdgMarg}
000631      PROCEDURE TLgHdngLayoutBox.MousePress(mouseLPT: LPoint);
000632          VAR layoutBox:              TLayoutBox;
000633              s:                      TListScanner;
000634              editLegendSelection:    TEditLegendSelection;
000635      BEGIN
000636          {$IFC fTrace}BP(11);{$ENDC}
000637          LRectHaveLPt(SELF.legendLayoutBox.extentLRect, mouseLPt);
000638          editLegendSelection := TEditLegendSelection(SELF.view.panel.selection.FreedAndReplacedBy(
000639                                              TEditLegendSelection.CREATE(NIL, SELF.Heap,
000640                                                  SELF.legendLayoutBox,
000641                                                  mouseLPT)));
000642          SELF.TabGrabbed; {get report on me right in the page status panel}
000643          editLegendSelection.textDialogImage.MousePress(mouseLPt);
000644          {$IFC fTrace}EP;{$ENDC}
000645      END;
000646
000647
000648  {$S HdgMarg}
000649      PROCEDURE TLgHdngLayoutBox.Move(deltaLPt: LPoint);
000650          VAR legendHeading:  TLegendHeading;
000651      BEGIN
000652          {$IFC fTrace}BP(10);{$ENDC}
000653          SUPERSELF.Move(deltaLPt);  {offsets and invalidates}
000654          legendHeading := TLegendHeading(SELF.manipulee);
000655          legendHeading.masterLegend.OffsetBy(deltaLPt);
000656    {$H-} LPtPlusLPt(legendHeading.offsetFromAlignment, deltaLPt, legendHeading.offsetFromAlignment);  {$H+}
000657      {$IFC fTrace}EP;{$ENDC}
000658      END;
000659
000660
000661  {$S HdgMarg}
000662      PROCEDURE TLgHdngLayoutBox.RecalcExtent;
000663          VAR newExtent:          LRect;
000664              oldExtent:          LRect;
000665              deltaLPt:           LPoint;
000666              newBaseLPoint:      LPoint;
000667              borders:            Rect;
```

```
000668              masterLegend:       TLegend;
000669              oldTopToBaseline:   LONGINT;
000670              newTopToBaseline:   LONGINT;
000671              legendHeading:      TLegendHeading;
000672              textExtent:         LRect;
000673              alignedToTop:       BOOLEAN;
000674              oldDescent:         INTEGER;
000675          PROCEDURE InvalOldAndNew;
000676              BEGIN
000677              thePad.InvalLRect(oldExtent);
000678              thePad.InvalLRect(newExtent);
000679              END;
000680      BEGIN
000681          {$IFC fTrace}BP(3);{$ENDC}
000682          oldExtent := SELF.extentLRect;
000683          legendHeading := TLegendHeading(SELF.manipulee);
000684          masterLegend := legendHeading.masterLegend;
000685
000686          borders := legendHeading.borders;
000687          oldTopToBaseline := legendHeading.topToBaseline;
000688
000689          SetParaExtent(masterLegend.paragraph, SELF.view, zeroLpt, textExtent);
000690          newTopToBaseline :=  - borders.top - textExtent.top;
000691
000692          alignedToTop := legendHeading.pageAlignment IN [aTopLeft, aTopCenter, aTopRight];
000693
000694          IF newTopToBaseline <> oldTopToBaseline THEN
000695              BEGIN
000696              IF alignedToTop THEN
000697                  legendHeading.offsetFromAlignment.v := legendHeading.offsetFromAlignment.v +
000698                      oldTopToBaseline - newTopToBaseline
000699              ELSE {bottom alignment}
000700                  BEGIN
000701                  WITH oldExtent DO
000702                      oldDescent := bottom - top - oldTopToBaseline;
000703                  legendHeading.offsetFromAlignment.v := legendHeading.offsetFromAlignment.v -
000704                      textExtent.bottom + oldDescent;
000705                  END;
000706              legendHeading.topToBaseline := newTopToBaseline;
000707              END;
000708
000709          LRectAddBorders(SELF.legendLayoutBox.extentLRect, borders, newExtent);
000710          legendHeading.Resize(newExtent);
000711
000712          legendHeading.LocateOnPage(TRUE);
000713          newExtent := legendHeading.extentLRect;
000714
000715          SetLPt(newBaseLPoint, newExtent.left - borders.left, newExtent.top + newTopToBaseline);
```

```
000716          masterLegend.location := newBaseLPoint;
000717          masterLegend.GetBoxRight;
000718          SELF.Resize(newExtent);
000719
000720          LPtMinusLPt(newExtent.topLeft, oldExtent.topLeft, deltaLPt);
000721          SELF.legendLayoutBox.OffsetLayoutBoxBy(deltaLPt, FALSE);   {its manipulee already ok}
000722
000723
000724          IF NOT equalLRect(oldExtent, newExtent) THEN
000725              SELF.view.panel.OnAllPadsDo(InvalOldAndNew);
000726
000727          SELF.TabGrabbed; {update page-status-dialog report panel}
000728          {$IFC fTrace}EP;{$ENDC}
000729      END;
000730
000731
000732
000733  {$S DlgInit}
000734  END;
000735
000736
000737  METHODS OF TPageStatusDialog;   {the status dialog in the PageDesign window}
000738
000739
000740  {$S DlgAlloc}
000741      FUNCTION TPageStatusDialog.CREATE(object: TObject; heap: THeap; itsPanel: TPanel): TPageStatusDialog;
000742          CONST
000743  {$IFC libraryVersion <= 20}
000744              spcAdjustment = -1;
000745  {$ELSEC}                            { S P R I N G }
000746              spcAdjustment = -1;
000747  {$ENDC}
000748          VAR cluster:        TCluster;
000749              aNumberString:  S255;
000750              inputFrame:     TInputFrame;
000751              button:         TButton;
000752              promptLoc:      LPoint;
000753              inputLoc:       LPoint;
000754              borders:        Rect;
000755              checkbox:       TCheckbox;
000756              newImage:       TDialogImage;
000757              typeStyle:      TTypeStyle;
000758              labelOffset:    Point;
000759              legend:         TLegend;
000760              boxSpacing:     INTEGER;
000761              itsID:          S255;
000762              itsLocation:    LPoint;
000763      BEGIN
```

```
000764          {$IFC fTrace}BP(11);{$ENDC}
000765          IF object = NIL THEN
000766              object := NewObject(heap, THISCLASS);
000767          SELF := TPageStatusDialog(TDialog.CREATE(object, heap, 'PAGE', itsPanel.view));
000768
000769          SELF.currentHeading := NIL;
000770
000771          MakeTypeStyle(famModern, size12Pitch, [], typeStyle);
000772
000773          SetPt(labelOffset, 6, 0);
000774          cluster := SELF.NewCluster(phOddEven);
000775          checkbox := cluster.NewCheckbox(phOddOnly, stdBoxWidth - 3, stdBoxHeight - 2,
000776              TRUE, labelOffset, typeStyle, FALSE);
000777          checkbox := cluster.NewAlignedCheckbox(phEvenOnly, FALSE);
000778          checkbox := cluster.NewAlignedCheckbox(phOddOrEven, TRUE);
000779          SELF.oddEvenCluster := cluster;
000780
000781          SetRect(borders, -18, -2, 18, 2);
000782          inputFrame := SELF.NewInputFrame(phMinPage, typeStyle, stdFrmeOffset, stdInputTypeStyle, 6,
000783               borders, FALSE, FALSE);
000784          LIntToStr(2, @aNumberString);
000785          inputFrame.SupplantContents(aNumberString);
000786          SELF.minPageFrame := inputFrame;
000787
000788          inputFrame := SELF.NewInputFrame(phMaxPage, typeStyle, stdFrmeOffset, stdInputTypeStyle, 6,
000789              borders, FALSE, FALSE);
000790          inputFrame.SupplantContents('------');
000791          SELF.maxPageFrame := inputFrame;
000792
000793          cluster := SELF.NewCluster(phAlignment);
000794          checkbox := cluster.NewCheckbox(phTopLeft, stdBoxWidth - 3, stdBoxHeight - 2, TRUE,
000795              labelOffset, typeStyle, FALSE);
000796          checkbox := cluster.NewAlignedCheckbox(phTopCenter, TRUE);
000797          checkbox := cluster.NewAlignedCheckbox(phTopRight, FALSE);
000798
000799          checkbox := cluster.NewCheckbox(phBotLeft, stdBoxWidth - 3, stdBoxHeight - 2, TRUE,
000800              labelOffset, typeStyle, FALSE); {for second row}
000801          checkbox := cluster.NewAlignedCheckbox(phBotCenter, FALSE);
000802          checkbox := cluster.NewAlignedCheckbox(phBotRight, FALSE);
000803          SELF.alignCluster := cluster;
000804
000805          button := SELF.NewButton(phLaunchHeading, stdButtonMetrics, NIL, NoCmdNumber);
000806          SELF.AddOKButton(noCmdNumber);
000807
000808          MakeTypeStyle(famModern, size10Pitch, [bold], typeStyle);
000809          legend := SELF.NewLegend(phPageMargins, typeStyle);
000810
000811          MakeTypeStyle(famModern, size12Pitch, [bold], typeStyle);
```

```
000812          cluster := SELF.NewCluster(phUnits);
000813          SELF.unitsCluster := cluster;
000814          checkbox := cluster.NewCheckbox(phInches, stdBoxWidth - 3, stdBoxHeight - 2, TRUE,
000815              labelOffset, typeStyle, TRUE);
000816          checkbox := cluster.NewAlignedCheckbox(phCentimeters, FALSE);
000817
000818          MakeTypeStyle(famModern, size15Pitch, [], typeStyle);
000819          legend := SELF.AddStdLegend('0.25  0.50  0.75  1.00  1.25  1.50  1.75  2.00  2.25  2.50',
000820                      96, 140, typeStyle);
000821          SELF.marginTitle := legend;
000822
000823          boxSpacing := stdBoxSpacing + spcAdjustment;
000824
000825          legend := SELF.NewLegend(phLeft, sysTypeStyle);
000826          cluster := SELF.NewRowOfBoxes(phLeftCluster, 10, 100, stdBoxWidth,
000827              stdBoxHeight, boxSpacing);
000828          cluster.selectBox(TCheckbox(cluster.ObjectWithIDNumber(103)));  {make this the real thing someday}
000829          SELF.leftCluster := cluster;
000830
000831          legend := SELF.NewLegend(phTop, sysTypeStyle);
000832          cluster := SELF.NewRowOfBoxes(phTopCluster, 10, 200, stdBoxWidth, stdBoxHeight, boxSpacing);
000833          cluster.selectBox(TCheckbox(cluster.ObjectWithIDNumber(203)));  {make this the real thing someday}
000834          SELF.topCluster := cluster;
000835
000836          legend := SELF.NewLegend(phRight, sysTypeStyle);
000837          cluster := SELF.NewRowOfBoxes(phRightCluster, 10, 300, stdBoxWidth, stdBoxHeight, boxSpacing);
000838          cluster.selectBox(TCheckbox(cluster.ObjectWithIDNumber(303)));  {make this the real thing someday}
000839          SELF.rightCluster := cluster;
000840
000841          legend := SELF.NewLegend(phBottom, sysTypeStyle);
000842          cluster := SELF.NewRowOfBoxes(phBotCluster, 10, 400, stdBoxWidth, stdBoxHeight, boxSpacing);
000843          cluster.selectBox(TCheckbox(cluster.ObjectWithIDNumber(403)));  {make this the real thing someday}
000844          SELF.bottomCluster := cluster;
000845
000846          button := SELF.NewButton(phInstallMargins, stdButtonMetrics, NIL, noCmdNumber);
000847          SELF.extentLRect.topLeft := zeroLPt; {+SW+}
000848          {$IFC fTrace}EP;{$ENDC}
000849      END;
000850
000851
000852      {$IFC fDebugMethods}
000853      {$S DlgDbg}
000854      PROCEDURE TPageStatusDialog.Fields(PROCEDURE Field(nameAndType: S255));
000855      BEGIN
000856          SUPERSELF.Fields(Field);
000857          Field('currentHeading: THeading');
000858          Field('oddEvenCluster: TCluster');
000859          Field('minPageFrame: TInputFrame');
```

```
000860          Field('maxPageFrame: TInputFrame');
000861          Field('alignCluster: TCluster');
000862          Field('unitsCluster: TCluster');
000863          Field('marginTitle: TLegend');
000864          Field('leftCluster: TCluster');
000865          Field('topCluster: TCluster');
000866          Field('rightCluster: TCluster');
000867          Field('bottomCluster: TCluster');
000868          Field('');
000869      END;
000870      {$ENDC}
000871
000872
000873  {$S HdgMarg}
000874      PROCEDURE TPageStatusDialog.ButtonPushed(button: TButton);
000875          VAR heading:        THeading;
000876              cluster:        TCluster;
000877              hitBoxID:       INTEGER;
000878              theS255:        S255;
000879              inputFrame:     TInputFrame;
000880              minPage:        LONGINT;
000881              maxPage:        LONGINT;
000882              pane:           TPane;
000883              pageDesignWindow:   TPageDesignWindow;
000884              plannerView:     TPlannerView;
000885              offset:         LPoint;
000886              layoutBox:      TLayoutBox;
000887              pageAlignment:  TPageAlignment;
000888              checkbox:       TCheckbox;
000889              oddOnly:        BOOLEAN;
000890              evenOnly:       BOOLEAN;
000891              newMargins:     LRect;
000892              panel:          TPanel;
000893              inches:         BOOLEAN;
000894              plannerPanel:   TPanel;
000895              editLegendSelection: TEditLegendSelection;
000896              noSelection:    TSelection;
000897          FUNCTION Margin(cluster: TCluster; baseID: INTEGER; vhs: vhSelect): INTEGER;
000898                  VAR hitBox: TCheckbox;
000899                      boxOrd: INTEGER;
000900              BEGIN
000901              hitBox := cluster.hiLitBox;
000902              IF hitBox = NIL THEN
000903                  boxOrd := 1
000904              ELSE
000905                  boxOrd := hitBox.idNumber - baseID + 1;
000906              IF inches THEN
000907                  Margin := (pageDesignWindow.hostView.res.vh[vhs] * boxOrd) DIV 4
```

```
000908              {because it's in quarter of inches right now}
000909          ELSE {operating in centimeters}
000910              Margin := LIntDivInt(pageDesignWindow.hostView.res.vh[vhs] * boxOrd * ORD4(100), 508);
000911          END;
000912      BEGIN
000913          {$IFC fTrace}BP(11);{$ENDC}
000914          pageDesignWindow := TPageDesignWindow(SELF.view.panel.window);
000915          plannerView := TPlannerView(pageDesignWindow.layoutPanel.view);
000916          IF button.idNumber = phLaunchHeading THEN  {launch a heading AND a corresponding new layout box}
000917              BEGIN
000918              offset := zeroLPt;  {default in case no ...}
000919
000920              cluster := SELF.alignCluster;
000921              IF cluster.hiLitBox = NIL THEN
000922                  cluster.SelectBox(TCheckbox(cluster.ObjectWithIDNumber(phTopCenter)));    {bulletproofing?}
000923
000924              SELF.InspectHeadingParms(oddOnly, evenOnly, pageAlignment, minPage, maxPage);
000925
000926              CASE pageAlignment OF
000927                  aTopLeft:       SetLPt(offset, 80, 30);
000928                  aTopCenter:     SetLPt(offset, 0, 30);
000929                  aTopRight:      SetLPt(offset, -80, 30);
000930                  aBottomLeft:    SetLPt(offset, 80, -30);
000931                  aBottomCenter:  SetLPt(offset, 0, -30);
000932                  aBottomRight:   SetLPt(offset, -80, -30);
000933                  END; {CASE}
000934
000935              IF minPage = maxPage THEN
000936                  IF odd(minPage) THEN
000937                      evenOnly := FALSE
000938                  ELSE
000939                      oddOnly := FALSE;  {keep user from launching a nowhere-printable heading}
000940
000941              heading := TLegendHeading.CREATE(NIL, SELF.Heap, pageDesignWindow.hostView.printManager,
000942                  dfltNewHeading, stdHdngTypeStyle, pageAlignment, offset, stdHdngBorders);
000943
000944              heading.minPage := minPage;
000945              heading.maxPage := maxPage;
000946
000947              heading.oddOnly := oddOnly;
000948              heading.evenOnly := evenOnly;
000949
000950              PushFocus;
000951              TPane(SELF.view.panel.panes.First).Focus;  {so that thePad will be set to something}
000952
000953              pageDesignWindow.hostView.printManager.headings.InsLast(heading);
000954              heading.AdjustForPage(0, TRUE);
000955              heading.LocateOnPage(TRUE);
```

```
000956
000957              SELF.currentHeading := heading;
000958              layoutBox := plannerView.NewLayoutBox(heading);
000959              IF layoutBox <> NIL THEN
000960                  BEGIN
000961                  plannerPanel := plannerView.panel;
000962                  plannerView.rootDialog.AddImage(layoutBox);
000963                  plannerPanel.BeginSelection;
000964                  editLegendSelection := TEditLegendSelection(plannerPanel.selection.FreedAndReplacedBy(
000965                                  TEditLegendSelection.CREATE(NIL, SELF.Heap,
000966                                  TLgHdngLayoutBox(layoutBox).legendLayoutBox, zeroLPt)));
000967                  editLegendSelection.coSelection.Become(
000968                          editLegendSelection.textDialogImage.textImage.text.SelectAll(
000969                          editLegendSelection.textDialogImage.textImage));
000970                  plannerPanel.InvalLRect(layoutBox.extentLRect);
000971                  plannerView.currentLayoutBox := layoutBox;
000972                  END;
000973
000974              TDialogView(SELF.view).AbandonThatButton; {turn off highlighting}
000975
000976              PopFocus;
000977
000978              currentWindow.PerformCommand(TCommand.CREATE(NIL, SELF.Heap, uCmdLaunchHeading, plannerView,
000979                  FALSE, revealAll));
000980              END
000981
000982          ELSE
000983
000984          IF button.idNumber = phInstallMargins THEN
000985              BEGIN
000986              inches := SELF.unitsCluster.hilitBox.idNumber = phInches;
000987              newMargins.left := Margin(SELF.leftCluster, 100, h);
000988              newMargins.top := Margin(SELF.topCluster, 200, v);
000989              newMargins.right := Margin(SELF.rightCluster, 300, h);
000990              newMargins.bottom := Margin(SELF.bottomCluster, 400, v);
000991              pageDesignWindow.hostView.printManager.ChangeMargins(newMargins);
000992              TDialogView(SELF.view).AbandonThatButton; {turn off highlighting}
000993              plannerView.panel.InvalLRect(plannerView.extentLRect);   {redraw layout panel with chgd margins}
000994              currentWindow.PerformCommand(TCommand.CREATE(NIL, SELF.Heap, uCmdInstallMargins, plannerView,
000995                  FALSE, revealNone));
000996              END
000997
000998          ELSE  {ok button}
000999
001000              BEGIN
001001              panel := plannerView.panel;
001002              panel.window.CommitLast;
001003              noSelection := panel.undoSelection.FreedAndReplacedBy(panel.view.NoSelection);
```

```
001004              panel.selection.Deselect;
001005              SUPERSELF.ButtonPushed(button);  {will give OK dismissal to dialog}
001006              END;
001007
001008          {$IFC fTrace}EP;{$ENDC}
001009      END;
001010
001011
001012  {$S HdgMarg}
001013      PROCEDURE TPageStatusDialog.InspectHeadingParms(VAR oddOnly, evenOnly: BOOLEAN;
001014                      VAR pageAlignment: TPageAlignment; VAR minPage, maxPage: LONGINT);
001015          VAR heading:            THeading;
001016              newPageAlignment:   TPageAlignment;
001017              theS255:            S255;
001018              checkbox:           TCheckbox;
001019              cState:             TConvResult;
001020      BEGIN
001021          {$IFC fTrace}BP(11);{$ENDC}
001022          checkbox := SELF.oddEvenCluster.hiLitBox;
001023
001024          IF checkbox = NIL THEN
001025              BEGIN
001026              oddOnly := FALSE;
001027              evenOnly := FALSE;
001028              END
001029          ELSE
001030              BEGIN
001031              oddOnly := (checkbox.idNumber = phOddOnly);
001032              evenOnly := (checkbox.idNumber = phEvenOnly);
001033              END;
001034
001035          checkbox := SELF.alignCluster.hiLitBox;
001036          IF checkbox = NIL THEN
001037              pageAlignment := aTopCenter
001038          ELSE
001039          CASE checkbox.idNumber OF
001040              phTopLeft:      pageAlignment := aTopLeft;
001041              phTopCenter:    pageAlignment := aTopCenter;
001042              phTopRight:     pageAlignment := aTopRight;
001043              phBotLeft:      pageAlignment := aBottomLeft;
001044              phBotCenter:    pageAlignment := aBottomCenter;
001045              phBotRight:     pageAlignment := aBottomRight;
001046              END;
001047
001048          SELF.maxPageFrame.GetContents(theS255);
001049          StrToLInt(@theS255, maxPage, cState);
001050          IF (cState <> cvValid) OR (maxPage <= 0) THEN
001051              BEGIN
```

```
001052              maxPage := MAXLINT;
001053              SELF.maxPageFrame.SupplantContents('------');
001054              END;
001055
001056          SELF.minPageFrame.GetContents(theS255);
001057          StrToLInt(@theS255, minPage, cState);
001058          IF (cState <> cvValid) OR (minPage > maxPage) THEN
001059              BEGIN
001060              minPage := 1;
001061              SELF.minPageFrame.SupplantContents('1');
001062              END;
001063
001064         {$IFC fTrace}EP;{$ENDC}
001065     END;
001066
001067
001068  {$S HdgMarg}
001069      PROCEDURE TPageStatusDialog.SetHeadingParms(oddOnly, evenOnly: BOOLEAN;
001070                              pageAlignment: TPageAlignment;  minPage, maxPage: LONGINT);
001071          VAR heading:            THeading;
001072              newPageAlignment:   TPageAlignment;
001073              theS255:            S255;
001074              checkbox:           TCheckbox;
001075              targetID:           INTEGER;
001076      BEGIN
001077          {$IFC fTrace}BP(11);{$ENDC}
001078          IF oddOnly THEN
001079              targetID := phOddOnly
001080          ELSE
001081          IF evenOnly THEN
001082              targetID := phEvenOnly
001083          ELSE
001084              targetID := phOddOrEven;
001085          SELF.oddEvenCluster.SelectBox(TCheckbox(SELF.oddEvenCluster.ObjectWithIDNumber(targetID)));
001086
001087          CASE pageAlignment OF
001088              aTopLeft:       targetID := phTopLeft;
001089              aTopCenter:     targetID := phTopCenter;
001090              aTopRight:      targetID := phTopRight;
001091              aBottomLeft:    targetID := phBotLeft;
001092              aBottomCenter:  targetID := phBotCenter;
001093              aBottomRight:   targetID := phBotRight;
001094              END;
001095
001096          SELF.alignCluster.SelectBox(TCheckbox(SELF.alignCluster.ObjectWithIDNumber(targetID)));
001097
001098          IntToStr(minPage, @theS255);
001099          SELF.minPageFrame.SupplantContents(theS255);
```

```
001100
001101          IF maxPage = maxLInt THEN
001102              theS255 := '------'
001103          ELSE
001104              IntToStr(maxPage, @theS255);
001105          SELF.maxPageFrame.SupplantContents(theS255);
001106         {$IFC fTrace}EP;{$ENDC}
001107      END;
001108
001109
001110  {$S HdgMarg}
001111      PROCEDURE TPageStatusDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN);
001112          VAR heading:            THeading;
001113              newPageAlignment:   TPageAlignment;
001114              newTitle:           S255;
001115              phIndex:            INTEGER;
001116              dummy:              LPoint;
001117      BEGIN
001118          {$IFC fTrace}BP(11);{$ENDC}
001119          IF checkBox.parent = SELF.unitsCluster THEN
001120              BEGIN
001121              IF checkbox.idNumber = phInches THEN
001122                  phIndex := phInchTitle
001123              ELSE
001124                  phIndex := phCmTitle;
001125              GetTextAndLocation(phIndex, newTitle, dummy);
001126              SELF.marginTitle.ChangeString(newTitle);
001127              END;
001128         {$IFC fTrace}EP;{$ENDC}
001129      END;
001130
001131
001132  {$S HdgMarg}
001133      FUNCTION TPageStatusDialog.DownAt(mouseLPt: LPoint): TDialogImage;
001134      BEGIN
001135          {$IFC fTrace}BP(11);{$ENDC}
001136          SELF.currentHeading := NIL;
001137          DownAt := SUPERSELF.DownAt(mouseLPt);
001138          {$IFC fTrace}EP;{$ENDC}
001139      END;
001140
001141
001142  {$S HdgMarg}
001143      PROCEDURE TPageStatusDialog.Draw;
001144          CONST   horizLine = 100;
001145                  vertLine = 570;
001146      BEGIN
001147          {$IFC fTrace}BP(11);{$ENDC}
```

```
001148          SUPERSELF.Draw; {draw the dialog}
001149          MoveToL(0, horizLine);
001150          PenNormal;
001151          PenSize(3, 2);
001152          LineToL(SELF.view.extentLRect.right, horizLine);
001153          MoveToL(vertLine, 0);
001154          LineToL(vertLine, horizLine);
001155         {$IFC fTrace}EP;{$ENDC}
001156      END;
001157
001158
001159
001160  {$S DlgInit}
001161  END;
001162
001163

End of File -- Lines: 1163 Characters: 41596
```

```
================================================================================================
FILE: "LIBTK/UDRAW.TEXT"
================================================================================================

000001  UNIT UDraw;
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004  {changed 05/01 1503 Changes to allow people to use Clascal on the Workshop}
000005
000006  {$Setc IsIntrinsic := TRUE }
000007
000008  {$IFC IsIntrinsic}
000009  INTRINSIC;
000010  {$ENDC}
000011
000012  INTERFACE
000013
000014  USES
000015      {$U UnitStd     } UnitStd,      {Client should not USE UnitStd}
000016      {$U UnitHz      } UnitHz,       {Client should not USE UnitHz and MUST NOT USE Storage}
000017      {$U libtk/UObject} UObject,           {Client must USE UObject}
000018      {$U -#BOOT-SysCall} SysCall,     {Client may USE SysCall}
000019  {$IFC LibraryVersion > 10}
000020      {$U LIBPL/PaslibCall} PaslibCall,
000021      {$U LIBPL/PPasLibc } PPasLibC,
000022  {$ENDC}
000023  {$IFC LibraryVersion <= 20}
000024      {$U FontMgr     } FontMgr,      {Client should USE UFont instead of FontMgr before QuickDraw}
000025  {$ENDC}
000026      {$U QuickDraw   } QuickDraw,    {Client must USE QuickDraw (unless we provide a type-stub for it)}
000027  {$IFC LibraryVersion > 20}
000028      {$U FontMgr     } FontMgr,      {Client should USE UFont instead of FontMgr after QuickDraw}
000029  {$ENDC}
000030      {$U WM.Events   } Events,
000031      {$U WM.Folders  } Folders,
000032      {$U FilerComm   } FilerComm;
000033
000034  {$SETC fDbgDraw     := fDbgOK}
000035  {$SETC fRngDraw     := fDbgOK}
000036  {$SETC fSymDraw     := fSymOK}
000037
000038  {$SETC fDebugMethods := fDbgDraw} {if VAR also true, trace entries and/or exits}
000039
000040  CONST
000041      {there should be at most 10 families and they should be in consecutive order; otherwise
000042              the command number constants in UABC should be changed}
000043      famSystem       =   0;
```

```
000044
000045      famMin          =   1;  {minimum family number that appears in the font menu}
000046      famModern       =   1;
000047      famClassic      =   2;
000048      famMax          =   2;
000049
000050      {there should be at most 20 families and they should be in consecutive order; otherwise
000051              the command number constants in UABC should be changed}
000052      sizeMin         =   1;
000053      size20Pitch     =   1;  { 8 Point 20 Pitch      NOTE: Modern available only}
000054      size15Pitch     =   2;  { 8 Point 15 Pitch      NOTE: Modern available only}
000055      size12Pitch     =   3;  {10 Point 12 Pitch}
000056      size10Pitch     =   4;  {12 Point 10 Pitch}
000057      size12Point     =   5;  {12 Point proportional}
000058      size14Point     =   6;  {14 Point proportional}
000059      size18Point     =   7;  {18 Point proportional}
000060      size24Point     =   8;  {24 Point proportional}
000061      sizeMax         =   8;
000062
000063      {font IDs to be used in QuickDraw}
000064      fIDSystem       =    0;     {Reserved for application generated text, that cannot be edited by user;
000065                                      does not print properly}
000066      fID20Pitch      =   19;
000067      fID15Pitch      =    7;
000068      fIDm12Pitch     =    8;
000069      fIDc12Pitch     =   13;
000070      fIDm10Pitch     =    9;
000071      fIDc10Pitch     =   14;
000072      fIDm12Point     =    4;
000073      fIDc12Point     =   10;
000074      fIDm14Point     =   15;
000075      fIDc14Point     =   16;
000076      fIDm18Point     =    5;
000077      fIDc18Point     =   11;
000078      fIDm24Point     =    6;
000079      fIDc24Point     =   12;
000080
000081      fIDRulers       =   25;     {Ruler Icons}
000082
000083      {fontIDs below this line are to be used only in special cases, there is no guarantee that these
000084              will print properly}
000085      fIDSysPatterns  =    2;     {System Patterns, ie. LisaDraw}
000086      fIDSysCursors   =    3;     {System Cursors}
000087      fIDLT20Graphics =   23;     {LisaTerminal 20 Pitch VT100 graphics}
000088      fIDLT12Graphics =   17;     {LisaTerminal 12 Pitch VT100 graphics}
000089      fIDLT20Text     =   27;     {LisaTerminal 20 Pitch VT100 text}
000090      fIDLT12Text     =   26;     {LisaTerminal 12 Pitch VT100 text}
000091      fIDDeskIcons    =   22;     {Desktop Icon font}
```

```
000092      fIDWM           =    1;     {Window Manager font}
000093      fIDCalculator   =   18;     {Calculator font}
000094      fIDIconName     =   21;     {Icon Name font}
000095      fIDMarker       =   20;     {Marker Font}
000096      fIDLisaGuide    =   24;     {LisaGuide Font}
000097
000098
000099  TYPE
000100
000101      TFontIDArray = ARRAY[famMin..famMax, sizeMin..sizeMax] OF INTEGER;
000102
000103      TScaler =
000104          RECORD   {scale-definition}
000105          numerator:      point; {numerator.h DIV denominator.h is the scale factor in horiz direction}
000106          denominator:    point; {numerator.v DIV denominator.v is the scale factor in the vert. direction}
000107          END;
000108
000109      TRectCoords = ARRAY[FALSE..TRUE] OF Point; {TRectCoords(aRect)[FALSE] = aRect.topLeft; [TRUE] = botRight}
000110
000111      LPoint =
000112              RECORD
000113                  CASE INTEGER OF
000114                      0: (v, h: LONGINT);
000115                      1: (vh: ARRAY [VHSelect] OF LONGINT)
000116              END;
000117
000118      LRect =
000119              RECORD
000120                  CASE INTEGER OF
000121                      0: (top, left, bottom, right: LONGINT);
000122                      1: (topLeft, botRight: LPoint)
000123              END;
000124
000125      LPattern = PACKED ARRAY[0..7] OF 0..255;
000126
000127      TLRectCoords = ARRAY[FALSE..TRUE] OF LPoint; {TLRectCoords(anLRect)[FALSE] = anLRect.topLeft; etc.}
000128
000129      TEnumActions =  (rErase, rFrame, rBackground, rDraw);
000130      TActions = SET OF TEnumActions;
000131
000132      THighTransit = (hNone, hOffToDim, hOffToOn, hDimToOn, hDimToOff, hOnToOff, hOnToDim);
000133      {Refresh assumes that the last four and only the last four start with already-highlighted stuff}
000134
000135      TEnumResizability = (userCanResizeIt, windowCanResizeIt);
000136      TResizability = SET OF TEnumResizability;       {arg for TBranchArea.CREATE & TPanel.Divide}
000137
000138      TFontRecord =
000139          PACKED RECORD
```

```
000140          CASE BOOLEAN OF
000141              FALSE:  (fontNum:       INTEGER);
000142              TRUE:   (fontFamily:    Byte;
000143                       fontSize:      Byte)
000144          END;
000145
000146      TTypeStyle =
000147          RECORD
000148  {$IFC LibraryVersion <= 20}
000149          onFaces:    TSeteface;
000150  {$ELSEC}
000151          onFaces:    Style;
000152  {$ENDC}
000153          font:       TFontRecord;
000154          END;
000155
000156      TArea = SUBCLASS OF TObject
000157
000158        {Variables}
000159          innerRect:      Rect;                       {window(usually)-relative bounds excluding borders}
000160          outerRect:      Rect;                       {bounding box in ancestral coordinates}
000161          parentBranch:   TBranchArea;                {only used for TPanels and TBranchAreas}
000162
000163        {Creation/Destruction}
000164          FUNCTION  TArea.CREATE(object: TObject; heap: THeap; itsRect: Rect): TArea; ABSTRACT;
000165
000166        {Attributes}
000167          FUNCTION  TArea.ChildWithPt(pt: Point; childList: TList; VAR nearestPt: Point): TArea;
000168          PROCEDURE TArea.GetBorder(VAR border: Rect); DEFAULT;
000169                                    {Return the deltas of the border bars, etc. (outer=inner+border)}
000170                                    {windows, bands, panes: 1 all around;
000171                                     panels: 1 on left/top, scroll bars on right/bottom}
000172          PROCEDURE TArea.GetMinExtent(VAR minExtent: Point; windowIsResizingIt: BOOLEAN); ABSTRACT;
000173          PROCEDURE TArea.SetOuterRect(newOuterRect: Rect);
000174          PROCEDURE TArea.SetInnerRect(newInnerRect: Rect);
000175
000176        {Display}                   {Other methods assume grafPort, origin, & clipping were preset by Focus}
000177          PROCEDURE TArea.Erase;
000178                                    {Erase the interior}
000179          PROCEDURE TArea.Focus; ABSTRACT;
000180                                    {Set up the grafPort for this window or pad}
000181          PROCEDURE TArea.Frame; DEFAULT;
000182                                    {Draw outlines, scroll bars, etc. outside the bounding box}
000183          PROCEDURE TArea.Refresh(rActions: TActions; highTransit: THighTransit); ABSTRACT;
000184
000185        {Buttoning}
000186          FUNCTION  TArea.DownAt(mousePt: Point): BOOLEAN; ABSTRACT;
000187
```

```
000188        {Resizing}
000189          PROCEDURE TArea.ResizeInside(newInnerRect: Rect); ABSTRACT;
000190          PROCEDURE TArea.ResizeOutside(newOuterRect: Rect); ABSTRACT;
000191
000192          END;
000193
000194
000195      TPad = SUBCLASS OF TArea
000196
000197        {Variables}
000198          port:           GrafPtr;                {the GrafPort used by this pad}
000199          viewedLRect:    LRect;                  {The portion of view that is displayed in innerRect}
000200          visLRect:       LRect;                  {viewedLRect sect visRgn while focused}
000201          availLRect:     LRect;                  {The larger part of view that fits in a 16-bit Rect}
000202          scrollOffset:   LPoint;                 {The distance scrolled from the view topLeft}
000203          origin:         Point;                  {What to set the grafport origin to when focused}
000204          cdOffset:       LPoint;                 {What to subtract from coordinates to get port coords}
000205          clippedRect:    rect;                   {additional clipping to apply at Focus time}
000206
000207          padRes:         Point;                  {spots/inch in the pad coordinate space}
000208          viewedRes:      Point;                  {spots/inch in the 32-bit space being projected}
000209
000210          scaled:         BOOLEAN;                {the net scale factor, combining zooming}
000211          scaleFactor:    TScaler;                {and aspect ratio, etc.}
000212
000213          zoomFactor:     TScaler;
000214
000215        {Creation/Destruction}
000216          FUNCTION  TPad.CREATE(object: TObject; heap: THeap; itsInnerRect: Rect; itsViewedLRect: LRect;
000217                                itsPadRes, itsViewRes: Point;
000218                                itsPort: GrafPtr): TPad;
000219
000220          PROCEDURE TPad.Redefine(itsInnerRect: Rect; itsViewedLRect: LRect;
000221                                  itsPadRes, itsViewRes: Point;
000222                             itsZoomFactor: TScaler; itsPort: GrafPtr);
000223
000224
000225        {Coordinate Mapping -- grafPort to view}
000226          PROCEDURE TPad.DistToLDist(distInPort: Point; VAR lDistInView: LPoint);
000227          PROCEDURE TPad.PatToLPat(patInPort: Pattern; VAR lPatInView: LPattern);
000228          PROCEDURE TPad.PtToLPt(ptInPort: Point; VAR lPtInView: LPoint);
000229          PROCEDURE TPad.RectToLRect(rectInPort: Rect; VAR lRectInView: LRect);
000230
000231        {Coordinate Mapping -- view to grafPort}
000232          PROCEDURE TPad.LDistToDist(lDistInView: LPoint; VAR distInPort: Point);
000233          PROCEDURE TPad.LPatToPat(lPatInView: LPattern; VAR patInPort: Pattern);
000234          PROCEDURE TPad.LPtToPt(lPtInView: LPoint; VAR ptInPort: Point);
000235          PROCEDURE TPad.LRectToRect(lRectInView: LRect; VAR rectInPort: Rect);
```

```
000236
000237        {Scrolling}
000238          PROCEDURE TPad.OffsetBy(deltaLPt: LPoint);   {offset viewedLRect -- no effect on display}
000239          PROCEDURE TPad.SetScrollOffset(VAR newOffset: LPoint);
000240                         {recalculates the origin and cdOffset fields; does not change arg}
000241
000242        {Display}
000243          PROCEDURE TPad.ClipFurtherTo(rBand: rect);  {narrows down clip area at next Focus}
000244          PROCEDURE TPad.Focus; OVERRIDE;
000245          PROCEDURE TPad.InvalLRect(r: LRect);              {Force redraw of r at next update}
000246          PROCEDURE TPad.InvalRect(r: Rect);                {Force redraw of r at next update}
000247          PROCEDURE TPad.SetPen(pen: PenState);    {NB: We should later augment this so that it scales
000248                                                       pensizes}
000249          PROCEDURE TPad.SetPenToHighlight(highTransit: THighTransit); {SetPenState to highlight this way}
000250
000251          PROCEDURE TPad.SetZoomFactor(zoomNumerator, zoomDenominator: point); DEFAULT;
000252
000253        {Drawing}
000254          PROCEDURE TPad.DrawLText(textBuf: Ptr; startByte, numBytes: INTEGER);
000255          PROCEDURE TPad.DrawLLine(newLPt: LPoint);
000256          PROCEDURE TPad.DrawLPicture(pic: PicHandle; r:LRect);
000257          PROCEDURE TPad.DrawLRect(verb: GrafVerb; r: LRect);
000258          PROCEDURE TPad.DrawLRRect(verb: GrafVerb; r: LRect; ovalWidth, ovalHeight: INTEGER);
000259          PROCEDURE TPad.DrawLOval(verb: GrafVerb; r: LRect);
000260          PROCEDURE TPad.DrawLArc(verb: GrafVerb; r: LRect; startAngle, arcAngle: INTEGER);
000261          PROCEDURE TPad.DrawLBits(VAR srcBits: BitMap; VAR srcRect: Rect;
000262                                   VAR dstLRect: LRect; mode: INTEGER; maskRgn: RgnHandle);
000263
000264        {Process termination and Debugging Assistance}
000265          PROCEDURE TPad.Crash; ABSTRACT;
000266          FUNCTION  TPad.BindHeap(activeVsClip, doBind: BOOLEAN): THeap; ABSTRACT;
000267
000268          END;
000269
000270      TBranchArea = SUBCLASS OF TArea
000271
000272        {Variables}
000273          arrangement:    VHSelect;       {v means above one another}
000274          elderFirst:     BOOLEAN;        {TRUE IFF elderChild is above or to the left of youngerChild}
000275          resizability:   TResizability;
000276          elderChild:     TArea;
000277          youngerChild:   TArea;
000278
000279        {Creation/Destruction}
000280          FUNCTION  TBranchArea.CREATE(object: TObject; heap: THeap; vhs: VHSelect; hasElderFirst: BOOLEAN;
000281                                       whoCanResizeIt: TResizability;
000282                                       itsElderChild, itsYoungerChild: TArea): TBranchArea;
000283
```

```
000284        {Attributes}
000285          PROCEDURE TBranchArea.GetMinExtent(VAR minExtent: Point; windowIsResizingIt: BOOLEAN); OVERRIDE;
000286          FUNCTION  TBranchArea.OtherChild(child: TArea): TArea;
000287          PROCEDURE TBranchArea.ReplaceChild(child, newChild: TArea);
000288          FUNCTION  TBranchArea.TopLeftChild: TArea;
000289
000290        {Resizing}
000291          PROCEDURE TBranchArea.ResizeOutside(newOuterRect: Rect); OVERRIDE;
000292          PROCEDURE TBranchArea.Redivide(newCd: INTEGER);
000293
000294          END;
000295
000296
000297
000298  VAR
000299
000300      amPrinting:         BOOLEAN;                    {Iff TRUE, we are currently printing rather than drawing}
000301
000302      zeroPt:             Point;                      {(0,0)}
000303      zeroRect:           Rect;                       {(0,0)-(0,0)}
000304      hugeRect:           Rect;                       {(0,0)-(MAXINT/2,MAXINT/2)}
000305
000306      zeroLPt:            LPoint;                     {(0,0)}
000307      zeroLRect:          LRect;                      {(0,0)-(0,0)}
000308      hugeLRect:          LRect;                      {(0,0)-(MAXLINT/2,MAXLINT/2)}
000309
000310      orthogonal:         ARRAY [v..h] OF VHSelect;   {Maps v to h and vice versa}
000311
000312      highPen:            ARRAY [THighTransit] OF PenState; {standard highlight-feedback transitions}
000313
000314      lPatWhite:          LPattern;                   {Maps to QuickDraw pattern white}
000315      lPatBlack:          LPattern;                   {Maps to QuickDraw pattern black}
000316      lPatGray:           LPattern;                   {Maps to QuickDraw pattern gray}
000317      lPatLtGray:         LPattern;                   {Maps to QuickDraw pattern ltGray}
000318      lPatDkGray:         LPattern;                   {Maps to QuickDraw pattern dkGray}
000319
000320      focusStack:         ARRAY [1..10] OF TArea;     {PushFocus pushes and PopFocus pops focusArea}
000321      focusStkPtr:        INTEGER;                    {Index of last thing on focusStack}
000322      focusArea:          TArea;                      {The currently focused area}
000323      focusRgn:           RgnHandle;                  {either padRgn or visRgn}
000324      padRgn:             RgnHandle;                  {intersection of pane and visRgn}
000325      altVisRgn:          RgnHandle;                  {If useAltVisRgn, use this instead of visRgn in Focus}
000326      useAltVisRgn:       BOOLEAN;                    {If TRUE, use altVisRgn instead of visRgn in Focus}
000327      thePad:             TPad;                       {focusArea, if a TPad, else NIL}
000328
000329      noPad:              TPad;                       {maps every point to itself}
000330      crashPad:           TPad;                       {an object willing to handle process termination}
000331
```

```
000332      screenRes:          Point;                      {screen resolution, pixels per inch}
000333
000334      sysTypeStyle:       TTypeStyle; {system font, normal face}
000335
000336      printerPseudoPort:  GrafPtr;
000337
000338      cArea:              TClass;
000339      cPad:               TClass;
000340      cBranchArea:        TClass;
000341
000342
000343
000344  {The next four are declared EXTERNAL in UOBJECT2}
000345  PROCEDURE InitQDWM;
000346  PROCEDURE TrmntExceptionHandler;
000347  PROCEDURE InitErrorAbort(error: INTEGER);
000348  {$IFC fDbgDraw}
000349  FUNCTION  BindHeap(activeVsClip, doBind: BOOLEAN): THeap;
000350  {$ENDC}
000351
000352  PROCEDURE Reduce(VAR numerator, denominator: INTEGER);   {reduce fraction to lowest terms}
000353
000354  FUNCTION  FPtPlusPt(operand1, operand2: Point): LONGINT;    {       p3 := Point(FPtPlusPt(p1, p2));       }
000355  FUNCTION  FPtMinusPt(operand1, operand2: Point): LONGINT;   { F stands for FUNCTION }
000356
000357  FUNCTION  FPtMulInt(operand1: Point; operand2: INTEGER): LONGINT;
000358  FUNCTION  FPtDivInt(operand1: Point; operand2: INTEGER): LONGINT;
000359              {e.g.:        center := Point(FPtDivInt(POINT(FPtPlusPt(p1, p2)), 2);          }
000360
000361  FUNCTION  FPtMaxPt(operand1, operand2: Point): LONGINT;     { each coordinate is max'ed separately }
000362  FUNCTION  FPtMinPt(operand1, operand2: Point): LONGINT;
000363
000364  FUNCTION  FDiagRect(operand1: Rect): LONGINT;               { FPtMinusPt(botRight-topLeft) }
000365
000366  PROCEDURE BoolToStr(bool: BOOLEAN; str: TPstring);
000367
000368  FUNCTION  LIntDivLInt(i, j: LONGINT): LONGINT;
000369  FUNCTION  LIntDivInt(i: LONGINT; j: INTEGER): LONGINT;
000370  FUNCTION  LIntMulInt(i: LONGINT; j: INTEGER): LONGINT;
000371
000372  FUNCTION  LIntOvrInt(i: LONGINT; j: INTEGER): LONGINT;
000373      {This returns LIntDivInt(i+(j DIV 2), j) if i>0 and
000374                    LIntDivInt(i-(j DIV 2), j) if i<0}
000375
000376  PROCEDURE PtPlusPt(operand1, operand2: Point; VAR result: Point);
000377  PROCEDURE PtMinusPt(operand1, operand2: Point; VAR result: Point);
000378  PROCEDURE PtMulInt(operand1: Point; operand2: INTEGER; VAR result: Point);
000379  PROCEDURE PtDivInt(operand1: Point; operand2: INTEGER; VAR result: Point);
```

```
000380  {$IFC LibraryVersion <= 20}
000381  FUNCTION  EqualPt(operand1, operand2: Point): BOOLEAN;  {Will be in QuickDraw eventually}
000382  {$ENDC}
000383
000384  PROCEDURE RectPlusRect(operand1, operand2: Rect; VAR result: Rect);
000385  PROCEDURE RectMinusRect(operand1, operand2: Rect; VAR result: Rect);
000386  {$IFC LibraryVersion <= 20}
000387  FUNCTION  EqualRect(rectA, rectB: Rect): BOOLEAN;       {Will be in QuickDraw eventually}
000388  FUNCTION  EmptyRect(r: Rect): BOOLEAN;                  {Will be in QuickDraw eventually}
000389  {$ENDC}
000390
000391  PROCEDURE AlignRect(VAR dstRect: Rect; srcRect: Rect; vhs: VHSelect);
000392  FUNCTION  LengthRect(r: Rect; vhs: VHSelect): INTEGER;
000393  FUNCTION  RectHasPt(dstRect: Rect; pt: Point): BOOLEAN;
000394  PROCEDURE RectHavePt(dstRect: Rect; VAR pt: Point);    {change pt so that topLeft <= pt <= botRight}
000395  FUNCTION  RectsNest(outer, inner: Rect): BOOLEAN;
000396  PROCEDURE RectifyRect(VAR dstRect: Rect);              {exchange coordinates until topLeft <= botRight}
000397  FUNCTION  RectIsVisible(rectInPort: Rect): BOOLEAN;
000398
000399  PROCEDURE PointToStr(pt: Point; str: TPstring);   {Referenced as EXTERNAL by UABC2}
000400  PROCEDURE RectToStr(r: Rect; str: TPstring);      {Referenced as EXTERNAL by UABC2}
000401
000402  PROCEDURE LPtPlusLPt(operand1, operand2: LPoint; VAR result: LPoint);
000403  PROCEDURE LPtMinusLPt(operand1, operand2: LPoint; VAR result: LPoint);
000404  PROCEDURE LPtMulInt(operand1: LPoint; operand2: INTEGER; VAR result: LPoint);
000405  PROCEDURE LPtDivInt(operand1: LPoint; operand2: INTEGER; VAR result: LPoint);
000406  FUNCTION  EqualLPt(operand1, operand2: LPoint): BOOLEAN;
000407
000408  PROCEDURE LRectPlusLRect(operand1, operand2: LRect; VAR result: LRect);
000409  PROCEDURE LRectMinusLRect(operand1, operand2: LRect; VAR result: LRect);
000410  FUNCTION  EqualLRect(rectA, rectB: LRect): BOOLEAN;
000411  FUNCTION  EmptyLRect(r: LRect): BOOLEAN;
000412
000413  PROCEDURE AlignLRect(VAR destLRect: LRect; srcLRect: LRect; vhs: VHSelect);
000414  FUNCTION  LengthLRect(r: LRect; vhs: VHSelect): LONGINT;
000415  FUNCTION  LRectHasLPt(destLRect: LRect; pt: LPoint): BOOLEAN;
000416  PROCEDURE LRectHaveLPt(destLRect: LRect; VAR pt: LPoint);    {change pt so that topLeft <= pt <= botRight}
000417  FUNCTION  LRectsNest(outer, inner: LRect): BOOLEAN;
000418  PROCEDURE RectifyLRect(VAR destLRect: LRect);              {exchange coordinates until topLeft <= botRight}
000419  FUNCTION  LRectIsVisible(srcLRect: LRect): BOOLEAN;
000420
000421  PROCEDURE LPointToStr(pt: LPoint; str: TPstring);   {Referenced as EXTERNAL by UOBJECT2}
000422  PROCEDURE LRectToStr(r: LRect; str: TPstring);      {Referenced as EXTERNAL by UOBJECT2}
000423
000424  PROCEDURE SetLPt(VAR destPt: LPoint; itsH, itsV: LONGINT);
000425  PROCEDURE SetLRect(VAR dstRect: LRect; itsLeft, itsTop, itsRight, itsBottom: LONGINT);
000426  PROCEDURE OffsetLRect(VAR dstRect: LRect; dh, dv: LONGINT);
000427  PROCEDURE InsetLRect(VAR dstRect: LRect; dh, dv: LONGINT);
```

```
000428  FUNCTION  SectLRect(srcRectA, srcRectB: LRect; VAR dstRect: LRect): BOOLEAN;
000429  PROCEDURE UnionLRect(srcRectA, srcRectB: LRect; VAR dstRect: LRect);
000430  FUNCTION  LPtInLRect(pt: LPoint; r: LRect): BOOLEAN;
000431
000432
000433  FUNCTION  IsSmallPt(srcPt: LPoint): BOOLEAN;
000434  FUNCTION  IsSmallRect(srcRect: LRect): BOOLEAN;
000435
000436
000437  (*PROCEDURE ClipLRect(r: LRect);  {Not yet implementable}*)
000438
000439
000440  {Drawing text}
000441
000442  PROCEDURE DrawLText(textBuf: Ptr; startByte, numBytes: INTEGER);
000443
000444  {Drawing lines, rectangles, and ovals}
000445
000446  PROCEDURE MoveToL(h, v: LONGINT);
000447  PROCEDURE MoveL(dh, dv: LONGINT);
000448  PROCEDURE LineToL(h, v: LONGINT);
000449  PROCEDURE LineL(dh, dv: LONGINT);
000450
000451  PROCEDURE FrameLRect(r: LRect);
000452  PROCEDURE PaintLRect(r: LRect);
000453  PROCEDURE EraseLRect(r: LRect);
000454  PROCEDURE InvrtLRect(r: LRect);
000455  PROCEDURE FillLRect(r: LRect; lPat: LPattern);
000456
000457  PROCEDURE FrameLOval(r: LRect);
000458  PROCEDURE PaintLOval(r: LRect);
000459  PROCEDURE EraseLOval(r: LRect);
000460  PROCEDURE InvrtLOval(r: LRect);
000461  PROCEDURE FillLOval(r: LRect; lPat: LPattern);
000462
000463  PROCEDURE FrameLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
000464  PROCEDURE PaintLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
000465  PROCEDURE EraseLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
000466  PROCEDURE InvrtLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
000467  PROCEDURE FillLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER; lPat: LPattern);
000468
000469  PROCEDURE FrameLArc(r: LRect; startAngle, arcAngle: INTEGER);
000470  PROCEDURE PaintLArc(r: LRect; startAngle, arcAngle: INTEGER);
000471  PROCEDURE EraseLArc(r: LRect; startAngle, arcAngle: INTEGER);
000472  PROCEDURE InvrtLArc(r: LRect; startAngle, arcAngle: INTEGER);
000473  PROCEDURE FillLArc(r: LRect; startAngle, arcAngle: INTEGER; lPat: LPattern);
000474
000475  FUNCTION  ClonePicture(pic: PicHandle; toHeap: THeap): PicHandle;
```

```
000476
000477  PROCEDURE ResizeFeedback(mousePt: Point; minPt, maxPt: Point; outerRect: Rect;
000478                           tabHeight, sbWidth, sbHeight: INTEGER; VAR newPt: Point);
000479
000480  PROCEDURE PushFocus;            {Save old focusArea on focusStack}
000481  PROCEDURE PopFocus;             {Restore old focusArea from focusStack and focus on it}
000482
000483  {$IFC LibraryVersion <= 20}
000484  PROCEDURE MakeTypeStyle(itsFamily: INTEGER; itsSize: INTEGER;
000485              itsFaces: TSetEFace;
000486              VAR typeStyle: TTypeStyle);
000487  {$ELSEC}
000488  PROCEDURE MakeTypeStyle(itsFamily: INTEGER; itsSize: INTEGER;
000489              itsFaces: Style;
000490              VAR typeStyle: TTypeStyle);
000491  {$ENDC}
000492
000493  FUNCTION  QDFontNumber(typeStyle: TTypeStyle): INTEGER;
000494  PROCEDURE SetQDTypeStyle(typeStyle: TTypeStyle);
000495
000496  IMPLEMENTATION
000497
000498  {$I libtk/UDRAW2.TEXT}
000499
000500  END.
000501
000502
000503
000504

End of File -- Lines: 504 Characters: 20727
```

```
================================================================================================
FILE: "LIBTK/UDRAW2.TEXT"
================================================================================================

000001
000002  {INCLUDE FILE UDRAW2 -- IMPLEMENTATION OF UDRAW}
000003  {Copyright 1983, 1984, Apple Computer, Inc.}
000004
000005
000006  {changed 05/01 1503 Changes to allow people to use Clascal on the Workshop}
000007
000008  {Segments: SgABCini(tialize and Terminate), SgDRWres(ident), SgABCc(o)ld, SgABCdbg}
000009
000010  {$IFC fRngDraw}
000011  {$R+}
000012  {$ELSEC}
000013  {$R-}
000014  {$ENDC}
000015
000016  {$IFC fSymDraw}
000017  {$D+}
000018  {$ELSEC}
000019  {$D-}
000020  {$ENDC}
000021
000022      CONST
000023          magicNumber     =   32768;
000024
000025      VAR fontID: TFontIDArray;
000026
000027  {$S SgDRWres}
000028
000029
000030  {$S SgABCini}
000031  PROCEDURE TrmntExceptionHandler;
000032      VAR ch:     CHAR;
000033          error:  INTEGER;
000034  BEGIN
000035      IF onDesktop THEN
000036          ImDying; {This must be done first}
000037
000038      IF NOT amDying THEN
000039          BEGIN
000040          {$IFC fDbgDraw}
000041          WriteLn('TrmntExceptionHandler');
000042          {$ENDC}
000043          amDying := TRUE;
```

```
000044          IF crashPad <> NIL THEN
000045              crashPad.Crash;
000046          END;
000047
000048      {$IFC fDbgDraw}
000049   {Flush the input queue in case there was user typeahead to the alternate screen}
000050      WHILE KeyPress DO
000051          Read(ch);
000052      {$ENDC}
000053
000054      IF NOT onDesktop THEN
000055          MoveConsole(error, mainscreen);
000056  END;
000057  {$S SgDRWres}
000058
000059
000060  {$S SgABCini}
000061  PROCEDURE QkDrError(error: INTEGER);
000062  BEGIN
000063      {$IFC fDbgDraw}
000064      ABCbreak('QkDrError', error);
000065      {$ENDC}
000066      HALT;
000067  END;
000068  {$S SgDRWres}
000069
000070
000071  {$S SgABCini}
000072  PROCEDURE InitQDWM;
000073      VAR error:          INTEGER;
000074          workDir:        Pathname;
000075          bootVol:        e_name;
000076          bootDir:        Pathname;
000077  {$IFC LibraryVersion < 30}
000078          bootPort:       tports;
000079  {$ENDC}
000080  BEGIN
000081      {$IFC libraryVersion <= 20}
000082      InitGraf(@thePort, @QkDrError);
000083      {$ELSEC}
000084      InitGraf(@thePort);
000085      {$ENDC}
000086
000087      crashPad := NIL;
000088
000089      IF onDesktop THEN
000090          BEGIN
000091          OpenWM;
```

```
000092          SetPort(deskPort);
000093          wmIsInitialized := TRUE;
000094          END
000095      ELSE
000096          BEGIN
000097          {move WriteLns to alternate screen}
000098          MoveConsole(error, alscreen);
000099  {$IFC fDbgDraw}
000100          IF error > 0 THEN
000101              ABCBreak('MoveConsole error', error);
000102  {$ENDC}
000103
000104          { set work directory to boot volume for FMOpen}
000105          Get_Working_Dir(error, workDir);
000106  {$IFC fDbgDraw}
000107          IF error > 0 THEN
000108              ABCBreak('Get_Working_Dir error', error);
000109  {$ENDC}
000110
000111  {$IFC LibraryVersion < 30}
000112          bootPort := OSBootVol(error);
000113  {$IFC fDbgDraw}
000114          IF error > 0 THEN
000115              ABCBreak('OSBootVol error', error);
000116  {$ENDC}
000117
000118          Get_Config_Name(error, bootPort, bootVol);
000119  {$IFC fDbgDraw}
000120          IF error > 0 THEN
000121              ABCBreak('Get_Config_Name error', error);
000122  {$ENDC}
000123  {$ELSEC}
000124          OSBootVol(error, bootVol);
000125  {$IFC fDbgDraw}
000126          IF error > 0 THEN
000127              ABCBreak('OSBootVol error', error);
000128  {$ENDC}
000129  {$ENDC}
000130          bootDir := CONCAT('-', bootVol);
000131
000132          Set_Working_Dir(error, bootDir);
000133  {$IFC fDbgDraw}
000134          IF error > 0 THEN
000135              ABCBreak('Set_Working_Dir to boot vol error', error);
000136  {$ENDC}
000137
000138          FMOpen(error);
000139  {$IFC fDbgDraw}
```

```
000140          IF error > 0 THEN
000141              ABCBreak('FMOpen error = ', error);
000142  {$ENDC}
000143
000144          { Set work directory back after OpenWM }
000145          Set_Working_Dir(error, workDir);
000146  {$IFC fDbgDraw}
000147          IF error > 0 THEN
000148              ABCBreak('Set_Working_Dir back to prefix error = ', error);
000149  {$ENDC}
000150          END;
000151  END;
000152  {$S SgDRWres}
000153
000154
000155  {$S SgABCdbg}
000156  FUNCTION  BindHeap(activeVsClip, doBind: BOOLEAN): THeap;
000157  BEGIN
000158      IF crashPad = NIL THEN
000159          BindHeap := NIL     {no UABC to do it for me}
000160      ELSE
000161          BindHeap := crashPad.BindHeap(activeVsClip, doBind);
000162  END;
000163  {$S SgDRWres}
000164
000165
000166  {$S SgABCcld}
000167  FUNCTION  FilerReason(error: INTEGER): FReason;
000168  BEGIN
000169          {$IFC fMaxTrace}BP(1);{$ENDC}
000170          {$IFC fMaxTrace}EP;{$ENDC}
000171      FilerReason := allOk;
000172      IF error > 0 THEN
000173          CASE error OF
000174               309:       FilerReason := noDiskSpace;
000175               315:       FilerReason := noMemory;
000176              4001:       FilerReason := badData;
000177              OTHERWISE   FilerReason := internalError;
000178              END;
000179  END;
000180  {$S SgDRWres}
000181
000182
000183  {$S SgABCini}
000184  PROCEDURE InitErrorAbort(error: INTEGER);
000185  BEGIN
000186      IF error > 0 THEN
000187          BEGIN
```

```
000188          {$IFC fDbgDraw}
000189          ABCbreak('InitErrorAbort', error);
000190          {$ENDC}
000191          IF onDesktop THEN
000192              TellFiler(error, initFailed, FilerReason(error), NIL);
000193          HALT;
000194          END
000195      ELSE
000196      IF wmIsInitialized THEN
000197          IF Abort THEN
000198              BEGIN
000199              IF onDesktop THEN
000200                  TellFiler(error, initFailed, aUserAbort, NIL);
000201              HALT;
000202              END;
000203  END;
000204  {$S SgDRWres}
000205
000206
000207  {$S SgDRWres}
000208  PROCEDURE Reduce(VAR numerator, denominator: INTEGER);      {reduce fraction to lowest terms}
000209      VAR factor: INTEGER;
000210          maxFactor: INTEGER; {also makes cosmetics}
000211          smallerNumerator:   INTEGER;
000212          smallerDenominator: INTEGER;
000213  BEGIN {very crude at the moment}
000214          {$IFC fMaxTrace}BP(1);{$ENDC}
000215          {$IFC fMaxTrace}EP;{$ENDC}
000216      maxFactor := MIN(numerator, denominator);
000217      FOR factor := maxFactor DOWNTO 2 DO
000218          BEGIN
000219          smallerNumerator := numerator DIV factor;
000220          smallerDenominator := denominator DIV factor;
000221          IF (factor * smallerNumerator = numerator) AND (factor * smallerDenominator = denominator) THEN
000222              BEGIN
000223              numerator := smallerNumerator;
000224              denominator := smallerDenominator;
000225              END;
000226          END;
000227  END;
000228
000229
000230  {$S SgDRWres}
000231  FUNCTION  FPtPlusPt(operand1, operand2: Point): LONGINT;
000232      VAR result: Point;
000233  BEGIN
000234          {$IFC fMaxTrace}BP(1);{$ENDC}
000235          {$IFC fMaxTrace}EP;{$ENDC}
```

```
000236      result.h := operand1.h + operand2.h;
000237      result.v := operand1.v + operand2.v;
000238      FPtPlusPt := LONGINT(result);
000239  END;
000240
000241
000242  {$S SgDRWres}
000243  FUNCTION  FPtMinusPt(operand1, operand2: Point): LONGINT;
000244      VAR result: Point;
000245  BEGIN
000246          {$IFC fMaxTrace}BP(1);{$ENDC}
000247          {$IFC fMaxTrace}EP;{$ENDC}
000248      result.h := operand1.h - operand2.h;
000249      result.v := operand1.v - operand2.v;
000250      FPtMinusPt := LONGINT(result);
000251  END;
000252
000253
000254  {$S SgABCdat}
000255  FUNCTION  FPtMulInt(operand1: Point; operand2: INTEGER): LONGINT;
000256      VAR result: Point;
000257  BEGIN
000258          {$IFC fMaxTrace}BP(1);{$ENDC}
000259          {$IFC fMaxTrace}EP;{$ENDC}
000260      result.h := operand1.h * operand2;
000261      result.v := operand1.v * operand2;
000262      FPtMulInt := LONGINT(result);
000263  END;
000264
000265
000266  {$S SgABCdat}
000267  FUNCTION  FPtDivInt(operand1: Point; operand2: INTEGER): LONGINT;
000268      VAR result: Point;
000269  BEGIN
000270          {$IFC fMaxTrace}BP(1);{$ENDC}
000271          {$IFC fMaxTrace}EP;{$ENDC}
000272      result.h := operand1.h DIV operand2;
000273      result.v := operand1.v DIV operand2;
000274      FPtDivInt := LONGINT(result);
000275  END;
000276
000277
000278  {$S SgDRWres}
000279  FUNCTION  FPtMaxPt(operand1, operand2: Point): LONGINT;
000280      VAR result: Point;
000281  BEGIN
000282          {$IFC fMaxTrace}BP(1);{$ENDC}
000283          {$IFC fMaxTrace}EP;{$ENDC}
```

```
000284      result.h := Max(operand1.h, operand2.h);
000285      result.v := Max(operand1.v, operand2.v);
000286      FPtMaxPt := LONGINT(result);
000287  END;
000288
000289
000290  {$S SgDRWres}
000291  FUNCTION  FPtMinPt(operand1, operand2: Point): LONGINT;
000292      VAR result: Point;
000293  BEGIN
000294          {$IFC fMaxTrace}BP(1);{$ENDC}
000295          {$IFC fMaxTrace}EP;{$ENDC}
000296      result.h := Min(operand1.h, operand2.h);
000297      result.v := Min(operand1.v, operand2.v);
000298      FPtMinPt := LONGINT(result);
000299  END;
000300
000301
000302  {$S SgDRWres}
000303  FUNCTION  FDiagRect(operand1: Rect): LONGINT;
000304      VAR result: Point;
000305  BEGIN
000306          {$IFC fMaxTrace}BP(1);{$ENDC}
000307          {$IFC fMaxTrace}EP;{$ENDC}
000308      result.h := operand1.right - operand1.left;
000309      result.v := operand1.bottom - operand1.top;
000310      FDiagRect := LONGINT(result);
000311  END;
000312
000313
000314  {$S SgABCdat}
000315  PROCEDURE BoolToStr(bool: BOOLEAN; str: TPstring);
000316  BEGIN
000317          {$IFC fMaxTrace}BP(1);{$ENDC}
000318          {$IFC fMaxTrace}EP;{$ENDC}
000319      IF bool THEN
000320          str^ := 'TRUE'
000321      ELSE
000322          str^ := 'FALSE';
000323  END;
000324
000325
000326  FUNCTION  LIntDivLInt(i, j: LONGINT): LONGINT;
000327      EXTERNAL;
000328
000329
000330  FUNCTION  LIntDivInt(i: LONGINT; j: INTEGER): LONGINT;
000331      EXTERNAL;
```

```
000332
000333
000334  FUNCTION  LIntMulInt(i: LONGINT; j: INTEGER): LONGINT;
000335      EXTERNAL;
000336
000337
000338  {$S SgDRWres}
000339  FUNCTION  LIntOvrInt(i: LONGINT; j: INTEGER): LONGINT;
000340  BEGIN
000341          {$IFC fMaxTrace}BP(1);{$ENDC}
000342          {$IFC fMaxTrace}EP;{$ENDC}
000343      IF i>0 THEN
000344          LIntOvrInt := LIntDivInt(i+(j DIV 2), j)
000345      ELSE
000346          LIntOvrInt := LIntDivInt(i-(j DIV 2), j);
000347  END;
000348
000349
000350  {$S SgABCdat}
000351  PROCEDURE PtPlusPt(operand1, operand2: Point; VAR result: Point);
000352  BEGIN
000353          {$IFC fMaxTrace}BP(1);{$ENDC}
000354          {$IFC fMaxTrace}EP;{$ENDC}
000355      result.h := operand1.h + operand2.h;
000356      result.v := operand1.v + operand2.v;
000357  END;
000358
000359
000360  {$S SgABCdat}
000361  PROCEDURE PtMinusPt(operand1, operand2: Point; VAR result: Point);
000362  BEGIN
000363          {$IFC fMaxTrace}BP(1);{$ENDC}
000364          {$IFC fMaxTrace}EP;{$ENDC}
000365      result.h := operand1.h - operand2.h;
000366      result.v := operand1.v - operand2.v;
000367  END;
000368
000369
000370  {$S SgABCdat}
000371  PROCEDURE PtMulInt(operand1: Point; operand2: INTEGER; VAR result: Point);
000372  BEGIN
000373          {$IFC fMaxTrace}BP(1);{$ENDC}
000374          {$IFC fMaxTrace}EP;{$ENDC}
000375      result.h := operand1.h * operand2;
000376      result.v := operand1.v * operand2;
000377  END;
000378
000379
```

```
000380  {$S SgABCdat}
000381  PROCEDURE PtDivInt(operand1: Point; operand2: INTEGER; VAR result: Point);
000382  BEGIN
000383          {$IFC fMaxTrace}BP(1);{$ENDC}
000384          {$IFC fMaxTrace}EP;{$ENDC}
000385      result.h := operand1.h DIV operand2;
000386      result.v := operand1.v DIV operand2;
000387  END;
000388
000389
000390  {$IFC LibraryVersion <= 20}
000391  FUNCTION  EqualPt(operand1, operand2: Point): BOOLEAN;
000392  BEGIN
000393      EqualPt := (operand1.h = operand2.h) AND (operand1.v = operand2.v);
000394  END;
000395  {$ENDC}
000396
000397
000398  {$S SgDRWres}
000399  PROCEDURE RectPlusRect(operand1, operand2: Rect; VAR result: Rect);
000400  BEGIN
000401          {$IFC fMaxTrace}BP(1);{$ENDC}
000402          {$IFC fMaxTrace}EP;{$ENDC}
000403      result.left := operand1.left + operand2.left;
000404      result.top := operand1.top + operand2.top;
000405      result.right := operand1.right + operand2.right;
000406      result.bottom := operand1.bottom + operand2.bottom;
000407  END;
000408
000409
000410  {$S SgDRWres}
000411  PROCEDURE RectMinusRect(operand1, operand2: Rect; VAR result: Rect);
000412  BEGIN
000413          {$IFC fMaxTrace}BP(1);{$ENDC}
000414          {$IFC fMaxTrace}EP;{$ENDC}
000415      result.left := operand1.left - operand2.left;
000416      result.top := operand1.top - operand2.top;
000417      result.right := operand1.right - operand2.right;
000418      result.bottom := operand1.bottom - operand2.bottom;
000419  END;
000420
000421
000422  {$IFC LibraryVersion <= 20}
000423  {$S SgDRWres}
000424  FUNCTION  EqualRect(rectA, rectB: Rect): BOOLEAN;
000425  BEGIN
000426          {$IFC fMaxTrace}BP(1);{$ENDC}
000427          {$IFC fMaxTrace}EP;{$ENDC}
```

```
000428      EqualRect := (rectA.left=rectB.left) AND (rectA.top=rectB.top) AND
000429                   (rectA.right=rectB.right) AND (rectA.bottom=rectB.bottom);
000430  END;
000431
000432
000433  {$S SgDRWres}
000434  FUNCTION  EmptyRect(r: Rect): BOOLEAN;
000435  BEGIN
000436          {$IFC fMaxTrace}BP(1);{$ENDC}
000437          {$IFC fMaxTrace}EP;{$ENDC}
000438      WITH r DO
000439          EmptyRect := (left >= right) OR (top >= bottom);
000440  END;
000441  {$ENDC}
000442
000443
000444  {$S SgDRWres}
000445  PROCEDURE AlignRect(VAR dstRect: Rect; srcRect: Rect; vhs: VHSelect);
000446  BEGIN
000447          {$IFC fMaxTrace}BP(1);{$ENDC}
000448          {$IFC fMaxTrace}EP;{$ENDC}
000449      dstRect.topLeft.vh[vhs] := srcRect.topLeft.vh[vhs];
000450      dstRect.botRight.vh[vhs] := srcRect.botRight.vh[vhs];
000451  END;
000452
000453
000454  {$S SgDRWres}
000455  FUNCTION  LengthRect(r: Rect; vhs: VHSelect): INTEGER;
000456  BEGIN
000457          {$IFC fMaxTrace}BP(1);{$ENDC}
000458          {$IFC fMaxTrace}EP;{$ENDC}
000459      LengthRect := r.botRight.vh[vhs] - r.topLeft.vh[vhs];
000460  END;
000461
000462
000463  {$S SgDRWres}
000464  FUNCTION  RectsNest(outer, inner: Rect): BOOLEAN;
000465  BEGIN
000466          {$IFC fMaxTrace}BP(1);{$ENDC}
000467          {$IFC fMaxTrace}EP;{$ENDC}
000468      RectsNest := RectHasPt(outer, inner.topLeft) AND RectHasPt(outer, inner.botRight);
000469  END;
000470
000471
000472  {$S SgDRWres}
000473  FUNCTION  RectHasPt(dstRect: Rect; pt: Point): BOOLEAN;
000474  BEGIN
000475          {$IFC fMaxTrace}BP(1);{$ENDC}
```

```
000476          {$IFC fMaxTrace}EP;{$ENDC}
000477      RectHasPt := (dstRect.left <= pt.h) AND (pt.h <= dstRect.right) AND
000478                   (dstRect.top <= pt.v) AND (pt.v <= dstRect.bottom);
000479  END;
000480
000481
000482  {$S SgDRWres}
000483  PROCEDURE RectHavePt(dstRect: Rect; VAR pt: Point);
000484  BEGIN {if dstRect is negative size, left/top is forced}
000485          {$IFC fMaxTrace}BP(1);{$ENDC}
000486          {$IFC fMaxTrace}EP;{$ENDC}
000487      pt.h := Max(dstRect.left, Min(dstRect.right, pt.h));
000488      pt.v := Max(dstRect.top, Min(dstRect.bottom, pt.v));
000489  END;
000490
000491
000492  {$S SgDRWres}
000493  PROCEDURE RectifyRect(VAR dstRect: Rect);
000494  BEGIN
000495          {$IFC fMaxTrace}BP(1);{$ENDC}
000496          {$IFC fMaxTrace}EP;{$ENDC}
000497      Pt2Rect(dstRect.topLeft, dstRect.botRight, dstRect);
000498  END;
000499
000500
000501  {$S SgDRWres}
000502  FUNCTION  RectIsVisible(rectInPort: Rect): BOOLEAN;
000503  BEGIN
000504          {$IFC fMaxTrace}BP(1);{$ENDC}
000505          {$IFC fMaxTrace}EP;{$ENDC}
000506      RectIsVisible := RectInRgn(rectInPort, focusRgn);
000507  END;
000508
000509
000510  {$S SgABCdbg}
000511  PROCEDURE PointToStr(pt: Point; str: TPstring);
000512      VAR s:  S255;
000513  BEGIN
000514          {$IFC fMaxTrace}BP(1);{$ENDC}
000515          {$IFC fMaxTrace}EP;{$ENDC}
000516      IntToStr(pt.h, str);
000517      IntToStr(pt.v, @s);
000518      str^ := CONCAT('(', str^, ',', s, ')');
000519  END;
000520
000521
000522  {$S SgABCdbg}
000523  PROCEDURE RectToStr(r: Rect; str: TPstring);
```

```
000524      VAR s:  S255;
000525  BEGIN
000526          {$IFC fMaxTrace}BP(1);{$ENDC}
000527          {$IFC fMaxTrace}EP;{$ENDC}
000528      PointToStr(r.topLeft, str);
000529      PointToStr(r.botRight, @s);
000530      str^ := CONCAT('[', str^, ',', s, ']');
000531  END;
000532  {$S SgDRWres}
000533
000534
000535  {$S SgDRWres}
000536  PROCEDURE LPtPlusLPt(operand1, operand2: LPoint; VAR result: LPoint);
000537  BEGIN
000538          {$IFC fMaxTrace}BP(1);{$ENDC}
000539          {$IFC fMaxTrace}EP;{$ENDC}
000540      result.h := operand1.h + operand2.h;
000541      result.v := operand1.v + operand2.v;
000542  END;
000543
000544
000545  {$S SgDRWres}
000546  PROCEDURE LPtMinusLPt(operand1, operand2: LPoint; VAR result: LPoint);
000547  BEGIN
000548          {$IFC fMaxTrace}BP(1);{$ENDC}
000549          {$IFC fMaxTrace}EP;{$ENDC}
000550      result.h := operand1.h - operand2.h;
000551      result.v := operand1.v - operand2.v;
000552  END;
000553
000554
000555  {$S SgABCdat}
000556  PROCEDURE LPtMulInt(operand1: LPoint; operand2: INTEGER; VAR result: LPoint);
000557  BEGIN
000558          {$IFC fMaxTrace}BP(1);{$ENDC}
000559          {$IFC fMaxTrace}EP;{$ENDC}
000560      result.h := operand1.h * operand2;
000561      result.v := operand1.v * operand2;
000562  END;
000563
000564
000565  {$S SgABCdat}
000566  PROCEDURE LPtDivInt(operand1: LPoint; operand2: INTEGER; VAR result: LPoint);
000567  BEGIN
000568          {$IFC fMaxTrace}BP(1);{$ENDC}
000569          {$IFC fMaxTrace}EP;{$ENDC}
000570      result.h := LIntDivInt(operand1.h, operand2);
000571      result.v := LIntDivInt(operand1.v, operand2);
```

```
000572  END;
000573
000574
000575  {$S SgDRWres}
000576  FUNCTION  EqualLPt(operand1, operand2: LPoint): BOOLEAN;
000577  BEGIN
000578          {$IFC fMaxTrace}BP(1);{$ENDC}
000579          {$IFC fMaxTrace}EP;{$ENDC}
000580      EqualLPt := (operand1.h = operand2.h) AND (operand1.v = operand2.v);
000581  END;
000582
000583
000584  {$S SgDRWres}
000585  PROCEDURE LRectPlusLRect(operand1, operand2: LRect; VAR result: LRect);
000586  BEGIN
000587          {$IFC fMaxTrace}BP(1);{$ENDC}
000588          {$IFC fMaxTrace}EP;{$ENDC}
000589      result.left := operand1.left + operand2.left;
000590      result.top := operand1.top + operand2.top;
000591      result.right := operand1.right + operand2.right;
000592      result.bottom := operand1.bottom + operand2.bottom;
000593  END;
000594
000595
000596  {$S SgDRWres}
000597  PROCEDURE LRectMinusLRect(operand1, operand2: LRect; VAR result: LRect);
000598  BEGIN
000599          {$IFC fMaxTrace}BP(1);{$ENDC}
000600          {$IFC fMaxTrace}EP;{$ENDC}
000601      result.left := operand1.left - operand2.left;
000602      result.top := operand1.top - operand2.top;
000603      result.right := operand1.right - operand2.right;
000604      result.bottom := operand1.bottom - operand2.bottom;
000605  END;
000606
000607
000608  {$S SgDRWres}
000609  FUNCTION  EqualLRect(rectA, rectB: LRect): BOOLEAN;
000610  BEGIN
000611          {$IFC fMaxTrace}BP(1);{$ENDC}
000612          {$IFC fMaxTrace}EP;{$ENDC}
000613      EqualLRect := (rectA.left=rectB.left) AND (rectA.top=rectB.top) AND
000614                    (rectA.right=rectB.right) AND (rectA.bottom=rectB.bottom);
000615  END;
000616
000617
000618  {$S SgDRWres}
000619  FUNCTION  EmptyLRect(r: LRect): BOOLEAN;
```

```
000620  BEGIN
000621          {$IFC fMaxTrace}BP(1);{$ENDC}
000622          {$IFC fMaxTrace}EP;{$ENDC}
000623      WITH r DO
000624          EmptyLRect := (left >= right) OR (top >= bottom);
000625  END;
000626
000627
000628  {$S SgDRWres}
000629  PROCEDURE AlignLRect(VAR destLRect: LRect; srcLRect: LRect; vhs: VHSelect);
000630  BEGIN
000631          {$IFC fMaxTrace}BP(1);{$ENDC}
000632          {$IFC fMaxTrace}EP;{$ENDC}
000633      destLRect.topLeft.vh[vhs] := srcLRect.topLeft.vh[vhs];
000634      destLRect.botRight.vh[vhs] := srcLRect.botRight.vh[vhs];
000635  END;
000636
000637
000638  {$S SgDRWres}
000639  FUNCTION  LengthLRect(r: LRect; vhs: VHSelect): LONGINT;
000640  BEGIN
000641          {$IFC fMaxTrace}BP(1);{$ENDC}
000642          {$IFC fMaxTrace}EP;{$ENDC}
000643      LengthLRect := r.botRight.vh[vhs] - r.topLeft.vh[vhs];
000644  END;
000645
000646
000647  {$S SgDRWres}
000648  FUNCTION  LRectsNest(outer, inner: LRect): BOOLEAN;
000649  BEGIN
000650          {$IFC fMaxTrace}BP(1);{$ENDC}
000651          {$IFC fMaxTrace}EP;{$ENDC}
000652      LRectsNest := LRectHasLPt(outer, inner.topLeft) AND LRectHasLPt(outer, inner.botRight);
000653  END;
000654
000655
000656  {$S SgDRWres}
000657  FUNCTION  LRectHasLPt(destLRect: LRect; pt: LPoint): BOOLEAN;
000658  BEGIN
000659          {$IFC fMaxTrace}BP(1);{$ENDC}
000660          {$IFC fMaxTrace}EP;{$ENDC}
000661      LRectHasLPt := (destLRect.left <= pt.h) AND (pt.h <= destLRect.right) AND
000662                      (destLRect.top <= pt.v) AND (pt.v <= destLRect.bottom);
000663  END;
000664
000665
000666  {$S SgDRWres}
000667  PROCEDURE LRectHaveLPt(destLRect: LRect; VAR pt: LPoint);
```

```
000668  BEGIN {if destLRect is negative size, left/top is forced}
000669          {$IFC fMaxTrace}BP(1);{$ENDC}
000670          {$IFC fMaxTrace}EP;{$ENDC}
000671      pt.h := Max(destLRect.left, Min(destLRect.right, pt.h));
000672      pt.v := Max(destLRect.top, Min(destLRect.bottom, pt.v));
000673  END;
000674
000675
000676  {$S SgDRWres}
000677  PROCEDURE RectifyLRect(VAR destLRect: LRect);
000678  BEGIN
000679          {$IFC fMaxTrace}BP(1);{$ENDC}
000680          {$IFC fMaxTrace}EP;{$ENDC}
000681      SetLRect(destLRect, Min(destLRect.left, destLRect.right), Min(destLRect.top, destLRect.bottom),
000682                          Max(destLRect.left, destLRect.right), Max(destLRect.top, destLRect.bottom));
000683  END;
000684
000685
000686  {$S SgDRWres}
000687  FUNCTION  LRectIsVisible(srcLRect: LRect): BOOLEAN;
000688      VAR rectInPort: Rect;
000689  BEGIN
000690          {$IFC fMaxTrace}BP(1);{$ENDC}
000691          {$IFC fMaxTrace}EP;{$ENDC}
000692      thePad.LRectToRect(srcLRect, rectInPort);
000693      IF EmptyRect(rectInPort) THEN
000694          LRectIsVisible := FALSE
000695      ELSE
000696          LRectIsVisible := RectInRgn(rectInPort, focusRgn);
000697  END;
000698
000699
000700  {$S SgABCdbg}
000701  PROCEDURE LPointToStr(pt: LPoint; str: TPstring);
000702      VAR s:  S255;
000703  BEGIN
000704          {$IFC fMaxTrace}BP(1);{$ENDC}
000705          {$IFC fMaxTrace}EP;{$ENDC}
000706      LIntToStr(pt.h, str);
000707      LIntToStr(pt.v, @s);
000708      str^ := CONCAT('(', str^, ',', s, ')');
000709  END;
000710
000711
000712  {$S SgABCdbg}
000713  PROCEDURE LRectToStr(r: LRect; str: TPstring);
000714      VAR s:  S255;
000715  BEGIN
```

```
000716          {$IFC fMaxTrace}BP(1);{$ENDC}
000717          {$IFC fMaxTrace}EP;{$ENDC}
000718      LPointToStr(r.topLeft, str);
000719      LPointToStr(r.botRight, @s);
000720      str^ := CONCAT('[', str^, ',', s, ']');
000721  END;
000722  {$S SgDRWres}
000723
000724
000725  {$S SgDRWres}
000726  PROCEDURE SetLPt(VAR destPt: LPoint; itsH, itsV: LONGINT);
000727  BEGIN
000728          {$IFC fMaxTrace}BP(1);{$ENDC}
000729          {$IFC fMaxTrace}EP;{$ENDC}
000730      WITH destPt DO
000731          BEGIN
000732          h := itsH;
000733          v := itsV;
000734          END;
000735  END;
000736
000737
000738  {$S SgDRWres}
000739  PROCEDURE SetLRect(VAR dstRect: LRect; itsLeft, itsTop, itsRight, itsBottom: LONGINT);
000740  BEGIN
000741          {$IFC fMaxTrace}BP(1);{$ENDC}
000742          {$IFC fMaxTrace}EP;{$ENDC}
000743      WITH dstRect DO
000744          BEGIN
000745          left := itsLeft;
000746          top := itsTop;
000747          right := itsRight;
000748          bottom := itsBottom;
000749          END;
000750  END;
000751
000752
000753  {$S SgDRWres}
000754  PROCEDURE OffsetLRect(VAR dstRect: LRect; dh, dv: LONGINT);
000755  BEGIN
000756          {$IFC fMaxTrace}BP(1);{$ENDC}
000757          {$IFC fMaxTrace}EP;{$ENDC}
000758      WITH dstRect DO
000759          BEGIN
000760          left := left + dh;
000761          top := top + dv;
000762          right := right + dh;
000763          bottom := bottom + dv;
```

```
000764          END;
000765  END;
000766
000767
000768  {$S SgDRWres}
000769  PROCEDURE InsetLRect(VAR dstRect: LRect; dh, dv: LONGINT);
000770  BEGIN
000771          {$IFC fMaxTrace}BP(1);{$ENDC}
000772          {$IFC fMaxTrace}EP;{$ENDC}
000773      WITH dstRect DO
000774          BEGIN
000775          left := left + dh;
000776          top := top + dv;
000777          right := right - dh;
000778          bottom := bottom - dv;
000779          IF (left >= right) OR (top >= bottom) THEN
000780              BEGIN
000781              left := 0;
000782              top := 0;
000783              right := 0;
000784              bottom := 0;
000785              END;
000786          END;
000787  END;
000788
000789
000790  {$S SgABCres}
000791  FUNCTION  SectLRect(srcRectA, srcRectB: LRect; VAR dstRect: LRect): BOOLEAN;
000792  BEGIN
000793          {$IFC fMaxTrace}BP(1);{$ENDC}
000794          {$IFC fMaxTrace}EP;{$ENDC}
000795      WITH dstRect DO
000796          BEGIN
000797          left := Max(srcRectA.left, srcRectB.left);
000798          top := Max(srcRectA.top, srcRectB.top);
000799          right := Min(srcRectA.right, srcRectB.right);
000800          bottom := Min(srcRectA.bottom, srcRectB.bottom);
000801          IF (left >= right) OR (top >= bottom) THEN
000802              BEGIN
000803              SectLRect := FALSE;
000804              left := 0;
000805              top := 0;
000806              right := 0;
000807              bottom := 0;
000808              END
000809          ELSE
000810              SectLRect := TRUE;
000811          END;
```

```
000812  END;
000813
000814
000815  {$S SgDRWres}
000816  PROCEDURE UnionLRect(srcRectA, srcRectB: LRect; VAR dstRect: LRect);
000817  BEGIN
000818          {$IFC fMaxTrace}BP(1);{$ENDC}
000819          {$IFC fMaxTrace}EP;{$ENDC}
000820      WITH dstRect DO
000821          BEGIN
000822          left := Min(srcRectA.left, srcRectB.left);
000823          top := Min(srcRectA.top, srcRectB.top);
000824          right := Max(srcRectA.right, srcRectB.right);
000825          bottom := Max(srcRectA.bottom, srcRectB.bottom);
000826          END;
000827  END;
000828
000829
000830  {$S SgDRWres}
000831  FUNCTION  LPtInLRect(pt: LPoint; r: LRect): BOOLEAN;
000832  BEGIN
000833          {$IFC fMaxTrace}BP(1);{$ENDC}
000834          {$IFC fMaxTrace}EP;{$ENDC}
000835      LPtInLRect := (r.left <= pt.h) AND (pt.h < r.right) AND
000836                    (r.top <= pt.v) AND (pt.v < r.bottom);
000837  END;
000838
000839
000840  {$S SgABCdat}
000841  FUNCTION  IsSmallPt(srcPt: LPoint): BOOLEAN;
000842  BEGIN
000843          {$IFC fMaxTrace}BP(1);{$ENDC}
000844          {$IFC fMaxTrace}EP;{$ENDC}
000845      IsSmallPt := (ABS(srcPt.h) < MAXINT) AND (ABS(srcPt.v) < MAXINT);
000846  END;
000847
000848
000849  {$S SgABCdat}
000850  FUNCTION  IsSmallRect(srcRect: LRect): BOOLEAN;
000851  BEGIN
000852          {$IFC fMaxTrace}BP(1);{$ENDC}
000853          {$IFC fMaxTrace}EP;{$ENDC}
000854      IsSmallRect := IsSmallPt(srcRect.topLeft) AND IsSmallPt(srcRect.botRight);
000855  END;
000856
000857
000858  {Drawing Text}
000859
```

```
000860  {$S SgABCdat}
000861  PROCEDURE DrawLText(textBuf: Ptr; startByte, numBytes: INTEGER);
000862  BEGIN
000863          {$IFC fMaxTrace}BP(1);{$ENDC}
000864          {$IFC fMaxTrace}EP;{$ENDC}
000865     {$IFC libraryVersion > 20}
000866      IF thePad.scaled THEN
000867          thePad.DrawLText(textBuf, startByte, numBytes)
000868      ELSE
000869          DrawText(QDPtr(textBuf), startByte, numBytes);
000870     {$ELSEC}
000871      DrawText(WordPtr(textBuf), startByte, numBytes);
000872     {$ENDC}
000873  END;
000874
000875
000876  {Drawing lines, rectangles, and ovals}
000877
000878  {$S SgDRWres}
000879  PROCEDURE MoveToL(h, v: LONGINT);
000880  VAR lPtInView:  LPoint;
000881      ptInPort:   Point;
000882  BEGIN
000883          {$IFC fMaxTrace}BP(1);{$ENDC}
000884          {$IFC fMaxTrace}EP;{$ENDC}
000885       SetLPt(lPtInView, h, v);
000886       thePad.LPtToPt(lPtInView, ptInPort);
000887       MoveTo(ptInPort.h, ptInPort.v);
000888  END;
000889
000890
000891  {$S SgDRWres}
000892  PROCEDURE MoveL(dh, dv: LONGINT);
000893  VAR lPtInView:  LPoint;
000894      ptInPort:   Point;
000895  BEGIN
000896          {$IFC fMaxTrace}BP(1);{$ENDC}
000897          {$IFC fMaxTrace}EP;{$ENDC}
000898       SetLPt(lPtInView, dh, dv);
000899       thePad.LDistToDist(lPtInView, ptInPort);
000900       Move(ptInPort.h, ptInPort.v);
000901  END;
000902
000903
000904  {$S SgDRWres}
000905  PROCEDURE LineToL(h, v: LONGINT);
000906  VAR lPtInView:     LPoint;
000907  BEGIN
```

```
000908          {$IFC fMaxTrace}BP(1);{$ENDC}
000909          {$IFC fMaxTrace}EP;{$ENDC}
000910       SetLPt(lPtInView, h, v);
000911       thePad.DrawLLine(lPtInView);
000912  END;
000913
000914
000915  {$S SgDRWres}
000916  PROCEDURE LineL(dh, dv: LONGINT);
000917  VAR lPtInView:  LPoint;
000918      ptInPort:   Point;
000919  BEGIN
000920          {$IFC fMaxTrace}BP(1);{$ENDC}
000921          {$IFC fMaxTrace}EP;{$ENDC}
000922       SetLPt(lPtInView, dh, dv);
000923       thePad.LDistToDist(lPtInView, ptInPort);
000924       Line(ptInPort.h, ptInPort.v);
000925  END;
000926
000927
000928  {$S SgDRWres}
000929  PROCEDURE FrameLRect(r: LRect);
000930  BEGIN
000931          {$IFC fMaxTrace}BP(1);{$ENDC}
000932          {$IFC fMaxTrace}EP;{$ENDC}
000933       thePad.DrawLRect(frame, r);
000934  END;
000935
000936
000937  {$S SgDRWres}
000938  PROCEDURE PaintLRect(r: LRect);
000939  BEGIN
000940          {$IFC fMaxTrace}BP(1);{$ENDC}
000941          {$IFC fMaxTrace}EP;{$ENDC}
000942       thePad.DrawLRect(paint, r);
000943  END;
000944
000945
000946  {$S SgDRWres}
000947  PROCEDURE EraseLRect(r: LRect);
000948  BEGIN
000949          {$IFC fMaxTrace}BP(1);{$ENDC}
000950          {$IFC fMaxTrace}EP;{$ENDC}
000951       thePad.DrawLRect(erase, r);
000952  END;
000953
000954
000955  {$S SgDRWres}
```

```
000956  PROCEDURE InvrtLRect(r: LRect);
000957  BEGIN
000958          {$IFC fMaxTrace}BP(1);{$ENDC}
000959          {$IFC fMaxTrace}EP;{$ENDC}
000960       thePad.DrawLRect(invert, r);
000961  END;
000962
000963
000964  {$S SgDRWres}
000965  PROCEDURE FillLRect(r: LRect; lPat: LPattern);
000966      VAR pat:    Pattern;
000967  BEGIN
000968          {$IFC fMaxTrace}BP(1);{$ENDC}
000969          {$IFC fMaxTrace}EP;{$ENDC}
000970      IF amPrinting THEN
000971          thePad.LPatToPat(lPat, pat);
000972  {$IFC LibraryVersion <= 20}
000973       thePat :=  Pattern(lPat);
000974  {$ELSEC}
000975       thePort^.fillPat :=  Pattern(lPat);
000976  {$ENDC}
000977       thePad.DrawLRect(fill, r);
000978  END;
000979
000980
000981  {$S SgDRWres}
000982  PROCEDURE FrameLOval(r: LRect);
000983  BEGIN
000984          {$IFC fMaxTrace}BP(1);{$ENDC}
000985          {$IFC fMaxTrace}EP;{$ENDC}
000986       thePad.DrawLOval(frame, r);
000987  END;
000988
000989
000990  {$S SgDRWres}
000991  PROCEDURE PaintLOval(r: LRect);
000992  BEGIN
000993          {$IFC fMaxTrace}BP(1);{$ENDC}
000994          {$IFC fMaxTrace}EP;{$ENDC}
000995       thePad.DrawLOval(paint, r);
000996  END;
000997
000998
000999  {$S SgDRWres}
001000  PROCEDURE EraseLOval(r: LRect);
001001  BEGIN
001002          {$IFC fMaxTrace}BP(1);{$ENDC}
001003          {$IFC fMaxTrace}EP;{$ENDC}
```

```
001004       thePad.DrawLOval(erase, r);
001005  END;
001006
001007
001008  {$S SgDRWres}
001009  PROCEDURE InvrtLOval(r: LRect);
001010  BEGIN
001011          {$IFC fMaxTrace}BP(1);{$ENDC}
001012          {$IFC fMaxTrace}EP;{$ENDC}
001013       thePad.DrawLOval(invert, r);
001014  END;
001015
001016
001017  {$S SgDRWres}
001018  PROCEDURE FillLOval(r: LRect; lPat: LPattern);
001019      VAR pat:    Pattern;
001020  BEGIN
001021          {$IFC fMaxTrace}BP(1);{$ENDC}
001022          {$IFC fMaxTrace}EP;{$ENDC}
001023      IF amPrinting THEN
001024          thePad.LPatToPat(lPat, pat);
001025  {$IFC LibraryVersion <= 20}
001026       thePat :=  Pattern(lPat);
001027  {$ELSEC}
001028       thePort^.fillPat :=  Pattern(lPat);
001029  {$ENDC}
001030       thePad.DrawLOval(fill, r);
001031  END;
001032
001033
001034  PROCEDURE FrameLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
001035  BEGIN
001036          {$IFC fMaxTrace}BP(1);{$ENDC}
001037          {$IFC fMaxTrace}EP;{$ENDC}
001038       thePad.DrawLRRect(frame, r, ovalWidth, ovalHeight);
001039  END;
001040
001041
001042  PROCEDURE PaintLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
001043  BEGIN
001044          {$IFC fMaxTrace}BP(1);{$ENDC}
001045          {$IFC fMaxTrace}EP;{$ENDC}
001046       thePad.DrawLRRect(paint, r, ovalWidth, ovalHeight);
001047  END;
001048
001049
001050  PROCEDURE EraseLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
001051  BEGIN
```

```
001052          {$IFC fMaxTrace}BP(1);{$ENDC}
001053          {$IFC fMaxTrace}EP;{$ENDC}
001054       thePad.DrawLRRect(erase, r, ovalWidth, ovalHeight);
001055  END;
001056
001057
001058  PROCEDURE InvrtLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER);
001059  BEGIN
001060          {$IFC fMaxTrace}BP(1);{$ENDC}
001061          {$IFC fMaxTrace}EP;{$ENDC}
001062       thePad.DrawLRRect(invert, r, ovalWidth, ovalHeight);
001063  END;
001064
001065
001066  PROCEDURE FillLRRect(r: LRect; ovalWidth, ovalHeight: INTEGER; lPat: LPattern);
001067      VAR pat:    Pattern;
001068  BEGIN
001069          {$IFC fMaxTrace}BP(1);{$ENDC}
001070          {$IFC fMaxTrace}EP;{$ENDC}
001071      IF amPrinting THEN
001072          thePad.LPatToPat(lPat, pat);
001073  {$IFC LibraryVersion <= 20}
001074       thePat :=  Pattern(lPat);
001075  {$ELSEC}
001076       thePort^.fillPat :=  Pattern(lPat);
001077  {$ENDC}
001078       thePad.DrawLRRect(fill, r, ovalWidth, ovalHeight)
001079  END;
001080
001081
001082  PROCEDURE FrameLArc(r: LRect; startAngle, arcAngle: INTEGER);
001083  BEGIN
001084          {$IFC fMaxTrace}BP(1);{$ENDC}
001085          {$IFC fMaxTrace}EP;{$ENDC}
001086       thePad.DrawLArc(frame, r, startAngle, arcAngle);
001087  END;
001088
001089
001090  PROCEDURE PaintLArc(r: LRect; startAngle, arcAngle: INTEGER);
001091  BEGIN
001092          {$IFC fMaxTrace}BP(1);{$ENDC}
001093          {$IFC fMaxTrace}EP;{$ENDC}
001094       thePad.DrawLArc(paint, r, startAngle, arcAngle);
001095  END;
001096
001097
001098  PROCEDURE EraseLArc(r: LRect; startAngle, arcAngle: INTEGER);
001099  BEGIN
```

```
001100          {$IFC fMaxTrace}BP(1);{$ENDC}
001101          {$IFC fMaxTrace}EP;{$ENDC}
001102       thePad.DrawLArc(erase, r, startAngle, arcAngle);
001103  END;
001104
001105
001106  PROCEDURE InvrtLArc(r: LRect; startAngle, arcAngle: INTEGER);
001107  BEGIN
001108          {$IFC fMaxTrace}BP(1);{$ENDC}
001109          {$IFC fMaxTrace}EP;{$ENDC}
001110       thePad.DrawLArc(invert, r, startAngle, arcAngle);
001111  END;
001112
001113
001114  PROCEDURE FillLArc(r: LRect; startAngle, arcAngle: INTEGER; lPat: LPattern);
001115      VAR pat:    Pattern;
001116  BEGIN
001117          {$IFC fMaxTrace}BP(1);{$ENDC}
001118          {$IFC fMaxTrace}EP;{$ENDC}
001119      IF amPrinting THEN
001120          thePad.LPatToPat(lPat, pat);
001121  {$IFC LibraryVersion <= 20}
001122       thePat :=  Pattern(lPat);
001123  {$ELSEC}
001124       thePort^.fillPat :=  Pattern(lPat);
001125  {$ENDC}
001126       thePad.DrawLArc(fill, r, startAngle, arcAngle);
001127  END;
001128
001129
001130  PROCEDURE RotatePattern(pInPat, pOutPat: Ptr; dh, dv: LONGINT);
001131      EXTERNAL;
001132
001133
001134  {$S SgABCdat}
001135  FUNCTION  ClonePicture(pic: PicHandle; toHeap: THeap): PicHandle;
001136      VAR h:  TH;
001137  BEGIN
001138          {$IFC fMaxTrace}BP(1);{$ENDC}
001139          {$IFC fMaxTrace}EP;{$ENDC}
001140      h := HAllocate(THz(toHeap), pic^^.picSize);
001141      XferLeft(Ptr(pic^), Ptr(h^), pic^^.picSize);
001142      ClonePicture := PicHandle(h);
001143  END;
001144  {$S SgDRWres}
001145
001146
001147  PROCEDURE ResizeFeedback(mousePt: Point; minPt, maxPt: Point; outerRect: Rect;
```

```
001148                          tabHeight, sbWidth, sbHeight: INTEGER; VAR newPt: Point);
001149
001150      VAR rFrame:     Rect;
001151          limitRect:  Rect;
001152          oldMousePt: Point;
001153          innerTop:   INTEGER;
001154          fTab:       BOOLEAN;
001155          fHscroll:   BOOLEAN;
001156          fVScroll:   BOOLEAN;
001157          event:      EventRecord;
001158          savePort:   GrafPtr;
001159
001160      PROCEDURE InitXorFrame;
001161      BEGIN
001162          fTab := TRUE;
001163          fHScroll := TRUE;
001164          fVScroll := TRUE;
001165
001166            { set up scroll bar and tab widths }
001167            { the +1 's are to account for enlarging rFrame by one pixel }
001168          IF sbWidth > 0 THEN
001169              sbWidth := sbWidth+1
001170          ELSE
001171              fVScroll := FALSE;
001172
001173          IF sbHeight > 0 THEN
001174              sbHeight := sbHeight+1
001175          ELSE
001176              fHScroll := FALSE;
001177
001178          IF tabHeight > 0 THEN
001179              tabHeight := tabHeight+1
001180          ELSE
001181              fTab := FALSE;
001182
001183            { setup rFrame - the outer rect for XORing  }
001184          rFrame := outerRect;
001185          InsetRect(rFrame, -1, -1);
001186
001187          limitRect.topLeft := minPt;
001188          limitRect.botRight := maxPt;
001189
001190          IF fTab THEN innerTop := rFrame.top+tabHeight
001191          ELSE innerTop := rFrame.top;
001192
001193            { Setup the pen }
001194          PenNormal;
001195          PenPat(gray);
```

```
001196          PenMode(notPatXor);
001197      END;
001198
001199      PROCEDURE XorFrame;
001200      BEGIN
001201          rFrame.botRight := newPt;
001202          FrameRect(rFrame);
001203          IF fTab THEN
001204              BEGIN
001205              MoveTo(rFrame.left, innerTop);
001206              LineTo(rFrame.right-1, innerTop);
001207              END;
001208          IF fHScroll THEN
001209              BEGIN
001210              MoveTo(rFrame.left, newPt.v-sbHeight);
001211              LineTo(rFrame.right-1, newPt.v-sbHeight);
001212              END;
001213          IF fVScroll THEN
001214              BEGIN
001215              MoveTo(newPt.h - sbWidth, innerTop);
001216              LineTo(newPt.h - sbWidth, rFrame.bottom-1);
001217              END;
001218      END;
001219
001220
001221      PROCEDURE DoDragFrame;
001222          VAR nxtPt:   Point;
001223      BEGIN
001224          nxtPt := Point(FPtPlusPt(newPt, Point(FPtMinusPt(mousePt, oldMousePt))));
001225          RectHavePt(limitRect, nxtPt);
001226          mousePt := Point(FPtPlusPt(Point(FPtMinusPt(nxtPt, newPt)), oldMousePt));
001227
001228          IF NOT EqualPt(nxtPt, newPt) THEN
001229              BEGIN
001230              XorFrame;  { hide old }
001231              newPt := nxtPt;
001232              XorFrame;  { draw new }
001233              END;
001234       END;
001235
001236  BEGIN
001237          {$IFC fMaxTrace}BP(1);{$ENDC}
001238          {$IFC fMaxTrace}EP;{$ENDC}
001239      InitXorFrame;  { sets rFrame }
001240
001241      newPt := rFrame.botRight;
001242      XorFrame;
001243
```

```
001244      oldMousePt := mousePt;
001245      WHILE StillDown DO
001246          BEGIN
001247          GetMouse(mousePt);
001248          DoDragFrame;
001249          oldMousePt := mousePt;
001250          END;
001251
001252      IF PeekEvent(event) AND (event.what = buttonUp) THEN
001253          BEGIN
001254          GetPort(savePort);
001255          SetPort(event.who);
001256          mousePt := event.where;
001257          LocalToGlobal(mousePt);
001258          SetPort(savePort);
001259          GlobalToLocal(mousePt);
001260          END
001261      ELSE
001262          GetMouse(mousePt);
001263
001264      DoDragFrame;
001265
001266      XorFrame;          { hide last }
001267      newPt.h := newPt.h - 1;
001268      newPt.v := newPt.v - 1;
001269  END; { ResizeFeedback }
001270
001271
001272  {$S SgABCres}
001273  PROCEDURE PopFocus;
001274  BEGIN
001275      {$IFC fTrace}BP(6);{$ENDC}
001276      SetEmptyRgn(padRgn); {To save memory space}
001277      focusArea := focusStack[focusStkPtr];
001278      thePad := NIL;
001279      IF focusArea <> NIL THEN
001280          focusArea.Focus;
001281      focusStkPtr := focusStkPtr - 1;
001282      {$IFC fTrace}EP;{$ENDC}
001283  END;
001284
001285
001286  {$S SgABCres}
001287  PROCEDURE PushFocus;
001288  BEGIN
001289      {$IFC fTrace}BP(6);{$ENDC}
001290      focusStkPtr := focusStkPtr + 1;
001291      focusStack[focusStkPtr] := focusArea;
```

```
001292      {$IFC fTrace}EP;{$ENDC}
001293  END;
001294
001295
001296  {$S SgDRWres}
001297  PROCEDURE MakeTypeStyle{(itsFamily: INTEGER; itsSize: INTEGER; itsFaces: TSetEFace/Style;
001298              VAR typeStyle: TTypeStyle)};
001299  BEGIN
001300      {$IFC fTrace}BP(11);{$ENDC}
001301      WITH typeStyle DO
001302          BEGIN
001303          onFaces := itsFaces;
001304          font.fontFamily := itsFamily;
001305          font.fontSize := itsSize;
001306          END;
001307      {$IFC fTrace}EP;{$ENDC}
001308  END;
001309
001310
001311  FUNCTION QDFontNumber{(typeStyle: TTypeStyle): INTEGER};
001312  BEGIN
001313      {$IFC fTrace}BP(11);{$ENDC}
001314      WITH typeStyle.font DO
001315          IF fontFamily = famSystem THEN
001316              QDFontNumber := fIDSystem
001317          ELSE
001318              QDFontNumber := fontID[fontFamily, fontSize];
001319      {$IFC fTrace}EP;{$ENDC}
001320  END;
001321
001322
001323  PROCEDURE SetQDTypeStyle{(typeStyle: TTypeStyle)};
001324  BEGIN
001325      {$IFC fTrace}BP(11);{$ENDC}
001326      TextFont(QDFontNumber(typeStyle));
001327      TextFace(typeStyle.onFaces);
001328      {$IFC fTrace}EP;{$ENDC}
001329  END;
001330
001331
001332  METHODS OF TArea;
001333
001334
001335      {$IFC fDebugMethods}
001336      {$S SgABCdbg}
001337      PROCEDURE TArea.Fields(PROCEDURE Field(nameAndType: S255));
001338      BEGIN
001339          Field('innerRect: Rect');
```

```
001340          Field('outerRect: Rect');
001341          Field('parentBranch: TBranchArea');
001342      END;
001343      {$S SgDRWres}
001344      {$ENDC}
001345
001346
001347      FUNCTION  TArea.ChildWithPt(pt: Point; childList: TList; VAR nearestPt: Point): TArea;
001348          VAR foundArea:  TArea;
001349              s:          TListScanner;
001350      BEGIN
001351          {$IFC fTrace}BP(6);{$ENDC}
001352          RectHavePt(SELF.innerRect, pt);
001353          s := childList.scanner;
001354          WHILE s.Scan(foundArea) DO
001355              IF RectHasPt(foundArea.outerRect, pt) THEN
001356                  s.Done;
001357          IF foundArea = NIL THEN
001358              BEGIN
001359              {$IFC fDbgDraw}
001360              ABCbreak('ChildWithPt found no area', 0);
001361              {$ENDC}
001362              foundArea := TArea(childList.First);
001363              END;
001364          RectHavePt(foundArea.innerRect, pt);
001365          nearestPt := pt;
001366          ChildWithPt := foundArea;
001367          {$IFC fTrace}EP;{$ENDC}
001368      END;
001369
001370
001371      PROCEDURE TArea.Erase;
001372      BEGIN
001373          {$IFC fTrace}BP(6);{$ENDC}
001374          FillRect(SELF.innerRect, white);
001375          {$IFC fTrace}EP;{$ENDC}
001376      END;
001377
001378
001379      PROCEDURE TArea.Frame;
001380          VAR innerRect:        Rect;
001381              borderRect:     Rect;
001382      BEGIN
001383          {$IFC fTrace}BP(6);{$ENDC}
001384          innerRect := SELF.innerRect;
001385          IF NOT RectsNest(innerRect, focusRgn^^.rgnBBox) THEN
001386              BEGIN
001387              PenNormal;
```

```
001388              PenSize(1, 1);
001389              borderRect := innerRect;
001390              InsetRect(borderRect, -1, -1);
001391              FrameRect(borderRect);
001392              END;
001393          {$IFC fTrace}EP;{$ENDC}
001394      END;
001395
001396
001397      PROCEDURE TArea.GetBorder(VAR border: Rect);
001398      BEGIN
001399          {$IFC fTrace}BP(3);{$ENDC}
001400          SetRect(border, -1, -1, 1, 1);
001401          {$IFC fTrace}EP;{$ENDC}
001402      END;
001403
001404
001405      PROCEDURE TArea.SetInnerRect(newInnerRect: Rect);
001406          VAR border: Rect;
001407      BEGIN
001408          {$IFC fTrace}BP(7);{$ENDC}
001409          SELF.innerRect := newInnerRect;
001410          SELF.GetBorder(border);
001411    {$H-} RectPlusRect(SELF.innerRect, border, SELF.outerRect); {$H+}
001412          {$IFC fTrace}EP;{$ENDC}
001413      END;
001414
001415
001416      PROCEDURE TArea.SetOuterRect(newOuterRect: Rect);
001417          VAR border: Rect;
001418      BEGIN
001419          {$IFC fTrace}BP(7);{$ENDC}
001420          SELF.outerRect := newOuterRect;
001421          SELF.GetBorder(border);
001422    {$H-} RectMinusRect(SELF.outerRect, border, SELF.innerRect); {$H+}
001423          {$IFC fTrace}EP;{$ENDC}
001424      END;
001425
001426
001427  {$S SgABCini}
001428  BEGIN
001429          fontID[famModern, size20Pitch] := fID20Pitch;
001430          fontID[famModern, size15Pitch] := fID15Pitch;
001431          fontID[famModern, size10Pitch] := fIDm10Pitch;
001432          fontID[famModern, size12Pitch] := fIDm12Pitch;
001433          fontID[famModern, size12Point] := fIDm12Point;
001434          fontID[famModern, size14Point] := fIDm14Point;
001435          fontID[famModern, size18Point] := fIDm18Point;
```

```
001436          fontID[famModern, size24Point] := fIDm24Point;
001437
001438          fontID[famClassic, size20Pitch] := fID20Pitch;
001439          fontID[famClassic, size15Pitch] := fID15Pitch;
001440          fontID[famClassic, size10Pitch] := fIDc10Pitch;
001441          fontID[famClassic, size12Pitch] := fIDc12Pitch;
001442          fontID[famClassic, size12Point] := fIDc12Point;
001443          fontID[famClassic, size14Point] := fIDc14Point;
001444          fontID[famClassic, size18Point] := fIDc18Point;
001445          fontID[famClassic, size24Point] := fIDc24Point;
001446
001447          MakeTypeStyle(famSystem, 0 {dummy}, [], sysTypeStyle);
001448  END;
001449  {$S SgDRWres}
001450
001451
001452  METHODS OF TPad;
001453
001454
001455      {$S sCldInit}
001456      FUNCTION  TPad.CREATE(object: TObject; heap: THeap; itsInnerRect: Rect; itsViewedLRect: LRect;
001457                            itsPadRes, itsViewRes: Point;
001458                            itsPort: GrafPtr): TPad;
001459          VAR zoomFactor: TScaler;
001460      BEGIN
001461          {$IFC fTrace}BP(7);{$ENDC}
001462          IF object = NIL THEN
001463              object := NewObject(heap, THISCLASS);
001464          SELF := TPad(object);
001465          SELF.parentBranch := NIL;
001466          SetPt(zoomFactor.numerator, 1, 1);
001467          SetPt(zoomFactor.denominator, 1, 1);
001468          SELF.Redefine(itsInnerRect, itsViewedLRect, itsPadRes, itsViewRes,
001469                        zoomFactor, itsPort);
001470          {$IFC fTrace}EP;{$ENDC}
001471      END;
001472      {$S SgDRWres}
001473
001474
001475      {$S sCldInit}
001476      PROCEDURE TPad.Redefine(itsInnerRect: Rect; itsViewedLRect: LRect;
001477                          itsPadRes, itsViewRes: Point;
001478                          itsZoomFactor: TScaler; itsPort: GrafPtr);
001479          VAR vhs:        VHSelect;
001480              newOffset:  LPoint;
001481      BEGIN
001482          {$IFC fTrace}BP(7);{$ENDC}
001483          SELF.SetInnerRect(itsInnerRect);
```

```
001484          WITH SELF, scaleFactor DO
001485       {$H-}  BEGIN
001486              port := itsPort;
001487              viewedLRect := itsViewedLRect;
001488              availLRect := itsViewedLRect;
001489              InsetLRect(availLRect, -8192, -8192);
001490              clippedRect := itsInnerRect;
001491              zoomFactor := itsZoomFactor;
001492
001493             {install new Resolutions}
001494              padRes := itsPadRes;
001495              viewedRes := itsViewRes;
001496
001497             {compute scale factor from resolutions and zoom factor}
001498              FOR vhs := v TO h DO
001499                  BEGIN
001500                  numerator.vh[vhs]   := itsPadRes.vh[vhs] * zoomFactor.numerator.vh[vhs];
001501                  denominator.vh[vhs] := itsViewRes.vh[vhs] * zoomFactor.denominator.vh[vhs];
001502                  Reduce(numerator.vh[vhs], denominator.vh[vhs]);
001503                  END;
001504
001505              scaled := (numerator.h <> denominator.h) OR (numerator.v <> denominator.v);
001506
001507             {compute scroll offset}
001508
001509              FOR vhs := v TO h DO
001510                  newOffset.vh[vhs] :=
001511                   LIntOvrInt(LIntMulInt(itsViewedLRect.topLeft.vh[vhs],
001512                                         numerator.vh[vhs]),
001513                              denominator.vh[vhs]) - itsInnerRect.topLeft.vh[vhs];
001514
001515              SELF.SetScrollOffset(newOffset);
001516       {$H+}  END;
001517          {$IFC fTrace}EP;{$ENDC}
001518       END;
001519       {$S SgDRWres}
001520
001521
001522       {$IFC fDebugMethods}
001523       {$S SgABCdbg}
001524       PROCEDURE TPad.Fields(PROCEDURE Field(nameAndType: S255));
001525       BEGIN
001526           TArea.Fields(Field);
001527           Field('port: Ptr');
001528           Field('viewedLRect: LRect');
001529           Field('visLRect: LRect');
001530           Field('availLRect: LRect');
001531           Field('scrollOffset: LPoint');
```

```
001532          Field('origin: Point'); {+LSR+}
001533          Field('cdOffset: LPoint'); {+LSR+}
001534          Field('clippedRect: rect');
001535          Field('padRes: Point');
001536          Field('viewedRes: Point');
001537          Field('scaled: BOOLEAN');
001538          Field('scaleFactor: RECORD numerator: Point; denominator: Point END');
001539          Field('zoomFactor: RECORD numerator: Point; denominator: Point END');
001540      END;
001541      {$S SgDRWres}
001542      {$ENDC}
001543
001544
001545      PROCEDURE TPad.ClipFurtherTo(rBand: rect);  {narrows down clip area at next Focus}
001546          VAR grafRect: Rect;
001547      BEGIN
001548          {$IFC fTrace}BP(7);{$ENDC}
001549     {$H-} IF SectRect(rBand, SELF.clippedRect, SELF.clippedRect) THEN; {$H+}
001550          {$IFC fTrace}EP;{$ENDC}
001551      END;
001552
001553
001554      PROCEDURE TPad.DistToLDist(distInPort: Point; VAR lDistInView: LPoint);
001555      BEGIN
001556          {$IFC fTrace}BP(6);{$ENDC}
001557          IF SELF.scaled THEN
001558              WITH SELF.scaleFactor DO
001559            {$H-} BEGIN
001560                  lDistInView.h := LIntOvrInt(LIntMulInt(distInPort.h, denominator.h), numerator.h);
001561                  lDistInView.v := LIntOvrInt(LIntMulInt(distInPort.v, denominator.v), numerator.v);
001562            {$H+} END
001563          ELSE
001564              BEGIN
001565              lDistInView.h := distInPort.h;
001566              lDistInView.v := distInPort.v;
001567              END;
001568          {$IFC fTrace}EP;{$ENDC}
001569      END;
001570
001571
001572      PROCEDURE TPad.DrawLLine(newLPt: LPoint);
001573      VAR newGrafPt: Point;
001574      BEGIN
001575          {$IFC fMaxTrace}BP(1);{$ENDC}
001576          {$IFC fMaxTrace}EP;{$ENDC}
001577          SELF.LPtToPt(newLPt, newGrafPt);
001578          StdLine(newGrafPt);
001579      END;
```

```
001580
001581
001582  {$IFC LibraryVersion <= 20}
001583  {This is still not the right implementation when we are printing}
001584      PROCEDURE TPad.DrawLPicture(pic: PicHandle; r:LRect);
001585          VAR rectInPort: Rect;
001586      BEGIN
001587          SELF.LRectToRect(r, rectInPort);
001588          DrawPicture(pic, rectInPort);
001589      END;
001590  {$ELSEC}
001591      PROCEDURE TKStdText(byteCount: INTEGER; textBuf: QDPtr; numer, denom: Point);
001592      BEGIN
001593          StdText(byteCount, textBuf, numer, numer);
001594      END;
001595
001596      PROCEDURE TKStdComment(kind, datasize: INTEGER; dataHandle: QDHandle);
001597      CONST
001598          picForeColor    =   108;
001599          picBackColor    =   109;
001600
001601      VAR pData:  TpLongint;
001602
001603      BEGIN
001604          IF dataHandle <> NIL THEN
001605              IF dataSize <> 4 THEN
001606                  BEGIN
001607                  pData := TpLongint(ORD(dataHandle^));
001608
001609                  CASE kind OF
001610                      picForeColor:
001611                          ForeColor(pData^);
001612                      picBackColor:
001613                          BackColor(pData^);
001614                      END;
001615                  END;
001616      END;
001617
001618  {This is still not the right implementation when we are printing}
001619      PROCEDURE TPad.DrawLPicture(pic: PicHandle; r:LRect);
001620          VAR rectInPort:     Rect;
001621              oldProcsPtr:    QDProcsPtr;
001622              TKProcs:        QDProcs;
001623              oldTextProc:    QDPtr;
001624              oldCommentProc: QDPtr;
001625      BEGIN
001626          {$IFC fMaxTrace}BP(1);{$ENDC}
001627          {$IFC fMaxTrace}EP;{$ENDC}
```

```
001628          WITH thePort^ DO
001629              BEGIN
001630              oldProcsPtr := grafprocs;
001631              IF oldProcsPtr = NIL THEN
001632                  BEGIN
001633                  SetStdProcs(TKProcs);
001634                  grafprocs := @TKProcs;
001635                  END;
001636              WITH grafprocs^ DO
001637                  BEGIN
001638                  oldTextProc := textProc;
001639                  oldCommentProc := commentProc;
001640                  IF amPrinting THEN
001641                      BEGIN
001642                      textProc := @TKStdText;
001643                      commentProc := @TKStdComment;
001644                      END;
001645                  END;
001646              END;
001647
001648          SELF.LRectToRect(r, rectInPort);
001649          DrawPicture(pic, rectInPort);
001650
001651          WITH thePort^ DO
001652              BEGIN
001653              IF oldProcsPtr <> NIL THEN
001654                  WITH grafprocs^ DO
001655                      BEGIN
001656                      textProcs := oldTextProc;
001657                      commentProc := oldCommentProc;
001658                      END;
001659              grafProcs := oldProcsPtr;
001660              END;
001661      END;
001662  {$ENDC}
001663
001664
001665      PROCEDURE TPad.DrawLRect(verb: GrafVerb; r: LRect);
001666      VAR rectInPort: Rect;
001667      BEGIN
001668          {$IFC fMaxTrace}BP(1);{$ENDC}
001669          {$IFC fMaxTrace}EP;{$ENDC}
001670          SELF.LRectToRect(r, rectInPort);
001671          StdRect(verb, rectInPort);
001672      END;
001673
001674
001675      PROCEDURE TPad.DrawLRRect(verb: GrafVerb; r: LRect; ovalWidth, ovalHeight: INTEGER);
```

```
001676      VAR rectInPort: Rect;
001677      BEGIN
001678          {$IFC fMaxTrace}BP(1);{$ENDC}
001679          {$IFC fMaxTrace}EP;{$ENDC}
001680          SELF.LRectToRect(r, rectInPort);
001681          StdRRect(verb, rectInPort, ovalWidth, ovalHeight);
001682      END;
001683
001684
001685      PROCEDURE TPad.DrawLOval(verb: GrafVerb; r: LRect);
001686      VAR rectInPort: Rect;
001687      BEGIN
001688          {$IFC fMaxTrace}BP(1);{$ENDC}
001689          {$IFC fMaxTrace}EP;{$ENDC}
001690          SELF.LRectToRect(r, rectInPort);
001691          StdOval(verb, rectInPort);
001692      END;
001693
001694
001695      PROCEDURE TPad.DrawLArc(verb: GrafVerb; r: LRect; startAngle, arcAngle: INTEGER);
001696      VAR rectInPort: Rect;
001697      BEGIN
001698          {$IFC fMaxTrace}BP(1);{$ENDC}
001699          {$IFC fMaxTrace}EP;{$ENDC}
001700          SELF.LRectToRect(r, rectInPort);
001701          StdArc(verb, rectInPort, startAngle, arcAngle);
001702      END;
001703
001704
001705      PROCEDURE TPad.DrawLBits(VAR srcBits: BitMap; VAR srcRect: Rect;
001706                               VAR dstLRect: LRect; mode: INTEGER; maskRgn: RgnHandle);
001707      VAR dstGrafRect: Rect;
001708      BEGIN
001709          {$IFC fMaxTrace}BP(1);{$ENDC}
001710          {$IFC fMaxTrace}EP;{$ENDC}
001711          SELF.LRectToRect(dstLRect, dstGrafRect);
001712          StdBits(srcBits, srcRect,dstGrafRect, mode, maskRgn);
001713      END;
001714
001715
001716  {$S SgABCres}
001717      PROCEDURE TPad.Focus;
001718          VAR visRgn: RgnHandle;
001719              origin: Point;
001720      BEGIN
001721          {$IFC fTrace}BP(6);{$ENDC}
001722          IF SELF.Port <> printerPseudoPort THEN
001723              SetPort(SELF.port);   {for the moment anyway don't tamper if being controlled by LisaPrint}
```

```
001724
001725          SetOrigin(0, 0);    {so thePort^.visRgn will be relative to a (0,0)-origined space, to match
001726                                  SELF.clippedRect and altVisRgn}
001727
001728          RectRgn(padRgn, SELF.clippedRect);
001729
001730          IF useAltVisRgn THEN
001731              visRgn := altVisRgn       {Instigated by TWindow.StashPicture or TClipboard.Publicize}
001732          ELSE
001733              visRgn := thePort^.visRgn;
001734
001735          SectRgn(padRgn, visRgn, padRgn);
001736
001737          origin := SELF.origin;
001738          WITH origin DO {+LSR+}
001739              BEGIN
001740              SetOrigin(h, v);
001741              OffsetRgn(padRgn, h, v);
001742              END;
001743
001744          SetClip(padRgn);
001745          focusRgn := padRgn;  {focusRgn is an alias for either padRgn or visRgn}
001746
001747          focusArea := SELF;
001748          thePad := SELF;
001749
001750          WITH SELF DO
001751        {$H-} BEGIN
001752              SELF.RectToLRect(focusRgn^^.rgnBBox, visLRect);
001753              IF SectLRect(viewedLRect, visLRect, visLRect) THEN
001754                  BEGIN END;
001755        {$H+} END;
001756          {$IFC fTrace}EP;{$ENDC}
001757      END;
001758
001759
001760      PROCEDURE TPad.InvalLRect(r: LRect);
001761          VAR rectInPort: Rect;
001762      BEGIN
001763          {$IFC fTrace}BP(7);{$ENDC}
001764          SELF.LRectToRect(r, rectInPort);
001765          SELF.InvalRect(rectInPort);
001766          {$IFC fTrace}EP;{$ENDC}
001767      END;
001768
001769
001770      PROCEDURE TPad.InvalRect(r: Rect);
001771      BEGIN
```

```
001772          {$IFC fTrace}BP(7);{$ENDC}
001773          IF SectRect(r, focusRgn^^.rgnBBox, r) THEN
001774              InvalRect(r);
001775          {$IFC fTrace}EP;{$ENDC}
001776      END;
001777  {$S SgDRWres}
001778
001779
001780      PROCEDURE TPad.LDistToDist(lDistInView: LPoint; VAR distInPort: Point);
001781      BEGIN
001782          {$IFC fTrace}BP(6);{$ENDC}
001783          IF SELF.scaled THEN
001784              WITH SELF.scaleFactor DO
001785            {$H-} BEGIN
001786                  distInPort.h := LIntOvrInt(LIntMulInt(lDistInView.h, numerator.h), denominator.h);
001787                  distInPort.v := LIntOvrInt(LIntMulInt(lDistInView.v, numerator.v), denominator.v);
001788            {$H+} END
001789          ELSE
001790              BEGIN
001791              distInPort.h := lDistInView.h;
001792              distInPort.v := lDistInView.v;
001793              END;
001794          {$IFC fTrace}EP;{$ENDC}
001795      END;
001796
001797
001798      PROCEDURE TPad.LPatToPat(lPatInView: LPattern; VAR patInPort: Pattern);
001799      BEGIN
001800          {$IFC fTrace}BP(6);{$ENDC}
001801          IF amPrinting THEN
001802              RotatePattern(@lPatInView, @patInPort, SELF.cdOffset.h, SELF.cdOffset.v)
001803          ELSE
001804              patInPort := Pattern(lPatInView); {+LSR+}
001805          {$IFC fTrace}EP;{$ENDC}
001806      END;
001807
001808
001809      PROCEDURE TPad.LPtToPt(lPtInView: LPoint; VAR ptInPort: Point);
001810      BEGIN
001811          {$IFC fTrace}BP(6);{$ENDC}
001812          LRectHaveLPt(SELF.availLRect, lPtInView);
001813          WITH SELF, cdOffset, scaleFactor DO {+LSR+}
001814              IF scaled THEN
001815            {$H-} BEGIN
001816                  ptInPort.h := LIntOvrInt(LIntMulInt(lPtInView.h, numerator.h), denominator.h) - h;
001817                  ptInPort.v := LIntOvrInt(LIntMulInt(lPtInView.v, numerator.v), denominator.v) - v;
001818            {$H+} END
001819              ELSE
```

```
001820                  BEGIN
001821                  ptInPort.h := lPtInView.h - h;
001822                  ptInPort.v := lPtInView.v - v;
001823                  END;
001824          {$IFC fTrace}EP;{$ENDC}
001825      END;
001826
001827
001828  {$S SgABCres}
001829      PROCEDURE TPad.LRectToRect(lRectInView: LRect; VAR rectInPort: Rect);
001830      BEGIN
001831          {$IFC fTrace}BP(6);{$ENDC}
001832          LRectHaveLPt(SELF.availLRect, lRectInView.topLeft);
001833          LRectHaveLPt(SELF.availLRect, lRectInView.botRight);
001834          WITH SELF, cdOffset, scaleFactor DO {+LSR+}
001835              IF scaled THEN
001836            {$H-} BEGIN
001837                  rectInPort.left := LIntOvrInt(LIntMulInt(lRectInView.left, numerator.h), denominator.h) - h;
001838                  rectInPort.top := LIntOvrInt(LIntMulInt(lRectInView.top, numerator.v), denominator.v) - v;
001839                  rectInPort.right := LIntOvrInt(LIntMulInt(lRectInView.right, numerator.h), denominator.h) - h;
001840                  rectInPort.bottom := LIntOvrInt(LIntMulInt(lRectInView.bottom, numerator.v), denominator.v)
001841                                       - v;
001842            {$H+} END
001843              ELSE
001844                  BEGIN
001845                  rectInPort.left := lRectInView.left - h;
001846                  rectInPort.top := lRectInView.top - v;
001847                  rectInPort.right := lRectInView.right - h;
001848                  rectInPort.bottom := lRectInView.bottom - v;
001849                  END;
001850          {$IFC fTrace}EP;{$ENDC}
001851      END;
001852  {$S SgDRWres}
001853
001854
001855      PROCEDURE TPad.OffsetBy(deltaLPt: LPoint);
001856          VAR vhs:        VHSelect;
001857              newOffset:  LPoint;
001858      BEGIN
001859          {$IFC fTrace}BP(7);{$ENDC}
001860          WITH SELF, deltaLPt DO
001861        {$H-} BEGIN
001862              OffsetLRect(viewedLRect, h, v);
001863              OffsetLRect(availLRect, h, v);
001864        {$H+} END;
001865
001866          FOR vhs := v TO h DO {$H-} {+LSR+}
001867              WITH SELF, scaleFactor DO
```

```
001868                newOffset.vh[vhs] := LIntOvrInt(LIntMulInt(viewedLRect.topLeft.vh[vhs],
001869                                                           numerator.vh[vhs]),
001870                                                denominator.vh[vhs]) - innerRect.topLeft.vh[vhs];
001871            {$H+}
001872            SELF.SetScrollOffset(newOffset);
001873            {$IFC fTrace}EP;{$ENDC}
001874        END;
001875
001876
001877        PROCEDURE TPad.PatToLPat(patInPort: Pattern; VAR lPatInView: LPattern);
001878        BEGIN
001879            {$IFC fTrace}BP(6);{$ENDC}
001880            IF amPrinting THEN
001881                RotatePattern(@patInPort, @lPatInView, -SELF.cdOffset.h, -SELF.cdOffset.v)
001882            ELSE
001883                LPatInView := LPattern(patInPort); {+LSR+}
001884            {$IFC fTrace}EP;{$ENDC}
001885        END;
001886
001887
001888        PROCEDURE TPad.PtToLPt(ptInPort: Point; VAR lPtInView: LPoint);
001889        {$IFC fDbgDraw}
001890        VAR pt: Point;
001891            s:  S255;
001892        {$ENDC}
001893        BEGIN
001894            {$IFC fTrace}BP(6);{$ENDC}
001895            WITH SELF, cdOffset, scaleFactor DO {+LSR+}
001896                IF scaled THEN
001897              {$H-} BEGIN
001898                    lPtInView.h := LIntOvrInt(LIntMulInt(ptInPort.h + h, denominator.h), numerator.h);
001899                    lPtInView.v := LIntOvrInt(LIntMulInt(ptInPort.v + v, denominator.v), numerator.v);
001900              {$H+} END
001901            ELSE
001902                BEGIN
001903                lPtInView.h := ptInPort.h + h;
001904                lPtInView.v := ptInPort.v + v;
001905                END;
001906            {$IFC fDbgDraw}
001907            SELF.LPtToPt(lPtInView, pt);
001908            IF NOT EqualPt(pt, ptInPort) THEN
001909                BEGIN
001910                PointToStr(ptInPort, @s);
001911                writeln('ptInPort:', s);
001912                LPointToStr(lPtInView, @s);
001913                writeln('lPtInView:',s);
001914                PointToStr(pt, @s);
001915                writeln('pt:', s);
```

```
001916              WrObj(SELF, 1, '');
001917              writeln;
001918              ABCbreak('Error in TPad.PtToLPt', 0);
001919              END;
001920          {$ENDC}
001921          {$IFC fTrace}EP;{$ENDC}
001922      END;
001923
001924
001925  {$S SgABCres}
001926      PROCEDURE TPad.RectToLRect(rectInPort: Rect; VAR lRectInView: LRect);
001927      BEGIN
001928          {$IFC fTrace}BP(6);{$ENDC}
001929          WITH SELF, cdOffset, scaleFactor DO {+LSR+}
001930              IF scaled THEN
001931            {$H-} BEGIN
001932                  lRectInView.left :=
001933                      LIntOvrInt(LIntMulInt(rectInPort.left + h, denominator.h), numerator.h);
001934                  lRectInView.top :=
001935                      LIntOvrInt(LIntMulInt(rectInPort.top + v, denominator.v), numerator.v);
001936                  lRectInView.right :=
001937                      LIntOvrInt(LIntMulInt(rectInPort.right + h, denominator.h), numerator.h);
001938                  lRectInView.bottom :=
001939                      LIntOvrInt(LIntMulInt(rectInPort.bottom + v, denominator.v), numerator.v);
001940            {$H+} END
001941          ELSE
001942              BEGIN
001943              lRectInView.left := rectInPort.left + h;
001944              lRectInView.top := rectInPort.top + v;
001945              lRectInView.right := rectInPort.right + h;
001946              lRectInView.bottom := rectInPort.bottom + v;
001947              END;
001948          {$IFC fTrace}EP;{$ENDC}
001949      END;
001950  {$S SgDRWres}
001951
001952
001953      PROCEDURE TPad.SetPen(pen: PenState);
001954          VAR  lPat:   LPattern;
001955      BEGIN
001956          {$IFC fTrace}BP(7);{$ENDC}
001957          IF amPrinting THEN
001958              BEGIN
001959              noPad.PatToLPat(pen.pnPat, lPat);
001960              SELF.LPatToPat(lPat, pen.pnPat);
001961              END;
001962          SetPenState(pen);
001963          {$IFC fTrace}EP;{$ENDC}
```

```
001964      END;
001965
001966
001967
001968      PROCEDURE TPad.SetPenToHighlight(highTransit: THighTransit);
001969      BEGIN
001970          {$IFC fTrace}BP(7);{$ENDC}
001971          SELF.SetPen(highPen[highTransit]);
001972          {$IFC fTrace}EP;{$ENDC}
001973      END;
001974
001975
001976      PROCEDURE TPad.SetScrollOffset(VAR newOffset: LPoint);
001977                          {recalculates the origin and cdOffset fields; does not change arg}
001978          VAR vhs:    VHSelect;
001979      BEGIN
001980          {$IFC fTrace}BP(7);{$ENDC}
001981          WITH SELF DO
001982              BEGIN
001983              scrollOffset := newOffset;
001984
001985              FOR vhs := v TO h DO
001986                  BEGIN
001987                  origin.vh[vhs] := newOffset.vh[vhs] MOD magicNumber;
001988                  cdOffset.vh[vhs] := newOffset.vh[vhs] - origin.vh[vhs];
001989                  END;
001990              END;
001991          {$IFC fTrace}EP;{$ENDC}
001992      END;
001993
001994
001995  {$S Override}
001996      PROCEDURE TPad.SetZoomFactor; {.... ONLY SEEMS TO BE RELEVANT FOR PANE--NONSENSE HERE FOR NOW}
001997      BEGIN
001998          {$IFC fTrace}BP(7);{$ENDC}
001999          {$IFC fTrace}EP;{$ENDC}
002000      END;
002001
002002
002003  {$S SgABCdat}
002004      PROCEDURE TPad.DrawLText(textBuf: Ptr; startByte, numBytes: INTEGER);
002005      BEGIN
002006          {$IFC fMaxTrace}BP(1);{$ENDC}
002007          {$IFC fMaxTrace}EP;{$ENDC}
002008          WITH SELF.zoomFactor DO {$H-}
002009     {$IFC libraryVersion > 20}
002010              StdText(numBytes, QDPtr(ORD(textBuf) + startByte), numerator, denominator);
002011     {$ELSEC}
```

```
002012              DrawText(WordPtr(textBuf), startByte, numBytes);
002013        {$ENDC}  {$H+}
002014         END;
002015  {$S SgDRWres}
002016
002017  {$S SgABCini}
002018  BEGIN
002019
002020      UnitAuthor('Apple');
002021      printerPseudoPort := POINTER(0);
002022      crashPad := NIL;
002023      SetPt(screenRes, 90, 60);
002024
002025      lPatWhite  := LPattern(white);
002026      lPatBlack  := LPattern(black);
002027      lPatGray   := LPattern(gray);
002028      lPatLtGray := LPattern(ltGray);
002029      lPatDkGray := LPattern(dkGray);
002030
002031      amPrinting := FALSE;
002032  END;
002033  {$S SgDRWres}
002034
002035
002036  METHODS OF TBranchArea;
002037
002038
002039      {$S SgABCcld}
002040      FUNCTION  TBranchArea.CREATE(object: TObject; heap: THeap; vhs: VHSelect; hasElderFirst: BOOLEAN;
002041                                   whoCanResizeIt: TResizability;
002042                                   itsElderChild, itsYoungerChild: TArea): TBranchArea;
002043      BEGIN
002044          {$IFC fTrace}BP(7);{$ENDC}
002045          IF object = NIL THEN
002046              object := NewObject(heap, THISCLASS);
002047          SELF := TBranchArea(object);
002048
002049          WITH SELF DO
002050              BEGIN
002051              outerRect := itsElderChild.outerRect;
002052              parentBranch := itsElderChild.parentBranch;
002053              arrangement := vhs;
002054              elderFirst := hasElderFirst;
002055              resizability := whoCanResizeIt;
002056              elderChild := itsElderChild;
002057              youngerChild := itsYoungerChild;
002058              END;
002059
```

```
002060          itsElderChild.parentBranch := SELF;
002061          itsYoungerChild.parentBranch := SELF;
002062          {$IFC fTrace}EP;{$ENDC}
002063      END;
002064      {$S SgDRWres}
002065
002066
002067      {$IFC fDebugMethods}
002068      {$S SgABCdbg}
002069      PROCEDURE TBranchArea.Fields(PROCEDURE Field(nameAndType: S255));
002070      BEGIN
002071          TArea.Fields(Field);
002072          Field('arrangement: Byte');
002073          Field('elderFirst: BOOLEAN');
002074          Field('resizability: Byte');
002075          Field('elderChild: TArea');
002076          Field('youngerChild: TArea');
002077      END;
002078      {$S SgDRWres}
002079      {$ENDC}
002080
002081
002082      {$S SgABCcld}
002083      PROCEDURE TBranchArea.GetMinExtent(VAR minExtent: Point; windowIsResizingIt: BOOLEAN);
002084          VAR elderMinSize:   Point;
002085              youngerMinSize: Point;
002086              vhs:            VHSelect;
002087      BEGIN
002088          {$IFC fTrace}BP(7);{$ENDC}
002089          vhs := SELF.arrangement;
002090
002091          SELF.elderChild.GetMinExtent(elderMinSize, TRUE);
002092          SELF.youngerChild.GetMinExtent(youngerMinSize, TRUE);
002093
002094          IF windowIsResizingIt AND NOT (windowCanResizeIt IN SELF.resizability) THEN
002095              youngerMinSize.vh[vhs] := LengthRect(SELF.youngerChild.outerRect, vhs);
002096
002097          minExtent.vh[vhs] := elderMinSize.vh[vhs] + youngerMinSize.vh[vhs];
002098
002099          vhs := orthogonal[vhs];
002100          minExtent.vh[vhs] := Max(elderMinSize.vh[vhs], youngerMinSize.vh[vhs]);
002101          {$IFC fTrace}EP;{$ENDC}
002102      END;
002103      {$S SgDRWres}
002104
002105
002106      {$S SgABCcld}
002107      FUNCTION  TBranchArea.OtherChild(child: TArea): TArea;
```

```
002108      BEGIN
002109          {$IFC fTrace}BP(7);{$ENDC}
002110          IF SELF.elderChild = child THEN
002111              OtherChild := SELF.youngerChild
002112          ELSE
002113          {$IFC fDbgDraw}
002114          IF SELF.youngerChild = child THEN
002115              OtherChild := SELF.elderChild
002116          ELSE
002117              ABCBreak('This panel branch does not have a child that is', ORD(child));
002118          {$ELSEC}
002119              OtherChild := SELF.elderChild;
002120          {$ENDC}
002121          {$IFC fTrace}EP;{$ENDC}
002122      END;
002123      {$S SgDRWres}
002124
002125
002126      {$S SgABCcld}
002127      PROCEDURE TBranchArea.Redivide(newCd: INTEGER);
002128          VAR elderRect:      Rect;
002129              youngerRect:    Rect;
002130      BEGIN
002131          {$IFC fTrace}BP(7);{$ENDC}
002132          elderRect := SELF.elderChild.outerRect;
002133          youngerRect := SELF.youngerChild.outerRect;
002134
002135          TRectCoords(elderRect)[SELF.elderFirst].vh[SELF.arrangement] := newCd;
002136          TRectCoords(youngerRect)[NOT SELF.elderFirst].vh[SELF.arrangement] := newCd;
002137
002138          SELF.elderChild.ResizeOutside(elderRect);
002139          SELF.youngerChild.ResizeOutside(youngerRect);
002140          {$IFC fTrace}EP;{$ENDC}
002141      END;
002142      {$S SgDRWres}
002143
002144
002145      {$S SgABCcld}
002146      PROCEDURE TBranchArea.ReplaceChild(child, newChild: TArea);
002147      BEGIN
002148          {$IFC fTrace}BP(7);{$ENDC}
002149          IF SELF.elderChild = child THEN
002150              SELF.elderChild := newChild
002151          ELSE
002152          {$IFC fDbgDraw}
002153          IF SELF.youngerChild = child THEN
002154              SELF.youngerChild := newChild
002155          ELSE
```

```
002156              ABCBreak('This panel branch does not have a child that is', ORD(child));
002157          {$ELSEC}
002158              SELF.youngerChild := newChild;
002159          {$ENDC}
002160
002161          newChild.parentBranch := SELF;
002162          IF child.parentBranch = SELF THEN
002163              child.parentBranch := NIL;
002164          {$IFC fTrace}EP;{$ENDC}
002165      END;
002166      {$S SgDRWres}
002167
002168
002169      {$S SgABCcld}
002170      PROCEDURE TBranchArea.ResizeOutside(newOuterRect: Rect);
002171          VAR formerRect:     Rect;
002172              elderChild:     TArea;
002173              youngerChild:   TArea;
002174              elderRect:      Rect;
002175              youngerRect:    Rect;
002176              vhs:            VHSelect;
002177              eldFirst:       BOOLEAN;
002178              minExtents:     ARRAY [FALSE..TRUE] OF Point;
002179              newCd:          INTEGER;
002180              deltaRect:      Rect;
002181      BEGIN
002182          {$IFC fTrace}BP(7);{$ENDC}
002183          formerRect := SELF.outerRect;
002184
002185          elderChild := SELF.elderChild;
002186          youngerChild := SELF.youngerChild;
002187
002188          elderRect := elderChild.outerRect;
002189          youngerRect := youngerChild.outerRect;
002190
002191          vhs := SELF.arrangement;
002192          eldFirst := SELF.elderFirst;
002193
002194          IF windowCanResizeIt IN SELF.resizability THEN
002195              BEGIN   {both children resize proportionally}
002196              MapRect(elderRect, formerRect, newOuterRect);
002197              MapRect(youngerRect, formerRect, newOuterRect);
002198
002199              elderChild.GetMinExtent(minExtents[NOT eldFirst], TRUE);
002200              youngerChild.GetMinExtent(minExtents[eldFirst], TRUE);
002201
002202              IF (minExtents[FALSE].vh[vhs] + minExtents[TRUE].vh[vhs]) < LengthRect(newOuterRect, vhs) THEN
002203                  BEGIN   {It is possible to satisfy both min constraints, so do so}
```

```
002204                  newCd := Max(newOuterRect.topLeft.vh[vhs] + minExtents[FALSE].vh[vhs],
002205                               Min(newOuterRect.botRight.vh[vhs] - minExtents[TRUE].vh[vhs],
002206                                   TRectCoords(elderRect)[eldFirst].vh[vhs]));
002207                  TRectCoords(elderRect)[eldFirst].vh[vhs] := newCd;
002208                  TRectCoords(youngerRect)[NOT eldFirst].vh[vhs] := newCd;
002209                  END;
002210              END
002211
002212          ELSE
002213              BEGIN   {only elder child resizes in my direction}
002214              RectMinusRect(newOuterRect, formerRect, deltaRect);
002215              RectPlusRect(elderRect, deltaRect, elderRect);
002216
002217              TRectCoords(deltaRect)[NOT eldFirst].vh[vhs] := TRectCoords(deltaRect)[eldFirst].vh[vhs];
002218              RectPlusRect(youngerRect, deltaRect, youngerRect);
002219              END;
002220
002221          youngerChild.ResizeOutside(youngerRect);
002222          elderChild.ResizeOutside(elderRect);
002223          SELF.outerRect := newOuterRect;
002224          {$IFC fTrace}EP;{$ENDC}
002225      END;
002226      {$S SgDRWres}
002227
002228
002229      {$S SgABCcld}
002230      FUNCTION  TBranchArea.TopLeftChild: TArea;
002231      BEGIN
002232          {$IFC fTrace}BP(7);{$ENDC}
002233          IF SELF.elderFirst THEN
002234              TopLeftChild := SELF.elderChild
002235          ELSE
002236              TopLeftChild := SELF.youngerChild;
002237          {$IFC fTrace}EP;{$ENDC}
002238      END;
002239      {$S SgDRWres}
002240
002241  {$S SgABCini}
002242  END;
002243  {$S SgDRWres}
002244  {$S SgABCini}
002245
002246
002247

End of File -- Lines: 2247 Characters: 58490
```

```
================================================================================================
FILE: "LIBTK/UOBJECT.TEXT"
================================================================================================

000001  UNIT UObject;
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003  {Implementation is in UOBJECT2-3-4}
000004
000005  {$SETC IsIntrinsic := TRUE}
000006
000007  {$IFC IsIntrinsic}
000008  INTRINSIC;
000009  {$ENDC}
000010
000011  {$SETC ErrsToFile := TRUE }
000012
000013  {$IFC ErrsToFile}
000014  {$E+}
000015  {*************************************}     {$E ERRS.TEXT}      {*************************************}
000016  {$ENDC}
000017
000018  {NOTE: The implementation of class TObject is quite obscure because this is actually system-type code}
000019
000020  {Segments: SgABCini(tialize), SgABCdat(a structures), SgABCdbg}
000021
000022  {
000023  ======================================== SPECIFICALLY IN UObject ========================================
000024
000025  ----------CLASSES------     -----------------VARIABLES------------------- ------------- COMMENTS ----------
000026
000027    TObject
000028
000029      TCollection             size dynOffset holeStart holeSize holeStd       -- indexed access (At, InsAt,
000030                                                                                  Each)
000031          TList                                                                 -- contains object handles
000032          TArray              recordBytes                                       -- contains records (even
000033                                                                                  lengths)
000034          TString                                                               -- contains characters
000035          TFile               path scanners                                     -- disk file (Exists, Rename)
000036      TScanner                collection position increment scanDone atEnd    -- sequential access (Scan,
000037                                                                                  Insert)
000038          TListScanner                                                          -- an object at a time
000039          TArrayScanner                                                         -- a record at a time
000040          TStringScanner      error actual                                      -- a character at a time (Xfer)
000041              TFileScanner    accesses refnum                                   -- through a whole TFile
000042
000043
```

```
000044  ======================================= IN ALL DATA STRUCTURE UNITS ======================================
000045
000046                 === KEY ===>   $ = in UObject  @ = in UHuge  * in UDb  # in UMac
000047
000048  ----------CLASSES----------     ---------------VARIABLES------------------ ------------ COMMENTS ----------
000049
000050  $ TObject
000051
000052  $   TCollection                   size dynOffset holeStart holeSize holeStd -- indexed access (At, InsAt,
000053                                                                                  Each)
000054  $       TList                                                                 -- contains object handles
000055    @         TLinkList               head tail                                   -- stored in TLinks
000056    @         THugeList               hugeArray                                   -- stored in linked blocks
000057  $       TArray                      recordBytes                               -- contains records (even
000058                                                                                    lengths)
000059    @         THugeArray              minBlockLength maxBlockLength blocks       -- impl. with linked blocks
000060  $       TString                                                                 -- contains characters
000061  $       TFile                       path scanners                             -- disk file (Exists, Rename)
000062    *     TDb                                                                   -- contains keyed records
000063    *         TDbFile                 file rScanDesc                              -- key is a PAOC/String
000064    *         TRsFile                 endIncrement firstKey lastKey scanners      -- key is a LONGINT (SwapIn)
000065    *             TDbRsFile           dbFile                                        -- implemented with a
000066                                                                                      TDbFile
000067          #       TMcRsFile           ???                                         -- implemented in the Mac
000068                                                                                      ROM
000069
000070  $   TScanner                    collection position increment scanDone atEnd -- sequential access (Scan,
000071                                                                                  Insert)
000072  $       TListScanner                                                          -- an object at a time
000073    @         TLnkLstScanner          scanLink
000074    @         THgeLstScanner          blkArrScanner
000075  $       TArrayScanner                                                         -- a record at a time
000076    @         THgeArrScanner          cacheBlock cacheIndex                       -- through a THugeArray
000077  $       TStringScanner              error actual                              -- a character at a time (Xfer)
000078  $           TFileScanner            accesses refnum                           -- through a whole TFile
000079    *         TRsScanner              whichWay key buffer                         -- through a single resource
000080    *     TDbScanner                  error                                     -- a key at a time
000081    *         TDbFiScanner            rScanDesc                                   -- through a TDbFile
000082    *         TRsFiScanner                                                        -- a resource at a time
000083    *             TDbRsFiScanner      dbRecSeq dbRecSize                            -- through a TDbRsFile
000084          #       TMcRsFiScanner      ???                                         -- implemented in the Mac
000085                                                                                      ROM
000086
000087    @ TLink                           element next                            -- has one element of a TLinkList
000088  }
000089
000090  INTERFACE
000091  {$SETC LibraryVersion := 30 } { 10 = 1.0 libraries; 13 = 1.3 libraries; 20 = Pepsi,
```

```
000092                                  29 = V12.0 Libraries, 30 = V13.0+ libraries }
000093  {$SETC compatibleLists := FALSE }
000094
000095  USES
000096
000097      UnitStd,
000098      UnitHz,
000099      {$U -#BOOT-SysCall   } SysCall,
000100  {$IFC LibraryVersion > 20}
000101      {$U LIBTK/Passwd  } Passwd,
000102  {$ENDC}
000103  {$IFC LibraryVersion <= 20}
000104      {$U UClascal}           UClascal,
000105  {$ELSEC} {$IFC LibraryVersion < 30}
000106      {$U LIBTK/UClascal}     UClascal,     {Needed for interface}
000107  {$ELSEC}
000108      {$U LIBPL/UClascal}     UClascal,     {Needed for interface}
000109  {$ENDC}
000110  {$ENDC}
000111        { The next units needed to find out where the printer is located, from parameter memory,
000112          so we can tell Paslib where it is. (Needed for debugger Output Redirect.) }
000113       PmDecl,
000114       Pmm,
000115  {$IFC LibraryVersion > 10}
000116       {$U LIBPL/PaslibCall} PaslibCall,
000117       {$U LIBPL/PPasLibc } PPasLibC,
000118  {$ENDC}
000119
000120       {$U HWInt}             HWInt;
000121
000122
000123  {$SETC fDbgOK := TRUE}{FALSE} {override UnitStd to test Tool Kit}
000124  {$SETC fSymOK := TRUE}{FALSE} {override UnitStd to test Tool Kit}
000125
000126  {$SETC fDbgObject    := fDbgOK}
000127  {$SETC fRngObject    := fDbgOK}
000128  {$SETC fSymObject    := fSymOK}
000129
000130  {$SETC fDebugMethods := fDbgObject} {include debugging methods in the compilation}
000131
000132  {$SETC fCheckHeap    := fDbgObject} {if VAR also true, check heap}
000133  {$SETC fTrace        := fDbgObject} {if VAR also true, trace entries/exits}
000134  {$SETC fMaxTrace     := fTrace AND FALSE} {if TRUE trace entries/exits on minor procedures too}
000135
000136  {$SETC fCheckIndices := fDbgObject} {if VAR also true, check subscripts}
000137
000138  CONST
000139
```

```
000140      prcsLdsn        = 1;            {ldsn for the process data segment}
000141      prcsDsBytes     = 15000;        {default heap size for a process data segment}
000142
000143      MaxBreaks       = 10;
000144
000145      outputRMargin   = 85;
000146      erInternal      = 4200;         {Stolen from list of errors in UABC for newHeap}
000147
000148      MAXLINT         = $7FFFFFFF;
000149
000150  TYPE
000151
000152    {Aliases needed to compile QuickDraw}
000153
000154      Ptr = ^LONGINT;
000155      ProcPtr = Ptr;
000156      Handle = ^Ptr;
000157
000158    {Aliases for commonly used types}
000159
000160      S8 = STRING[8];
000161      S255 = STRING[255];
000162
000163      TFilePath = S255;   {Increased from 66 because of the new hierarchical file system;
000164                               corresponds to Pathname in SYSCALL}
000165      TFilePart = STRING[32]; {length of each level in a pathname; corresponds to e_name in SYSCALL}
000166      TPassword = TFilePart;
000167
000168      THeap = Ptr;    {alias for THz in UnitHz}
000169      TClass = Ptr;   {alias for TPSliceTable in UClascal}
000170
000171      Byte = -128..127;
000172      TPString = ^S255;
000173
000174      TpINTEGER = ^INTEGER;
000175      TpLONGINT = ^LONGINT;
000176
000177      TAuthorName = STRING[32];
000178      TClassName  = STRING[8];
000179
000180      TClassWorld = RECORD    {Alias for TWorld in IMPLEMENTATION}
000181          infRecs:    TArray {OF name, size, author, & version information};
000182          classes:    TArray {OF TClass -- the pointer in each Clascal object};
000183          authors:    TArray {OF PACKED ARRAY [1..SIZEOF(TAuthorName)] OF CHAR};
000184          aliases:    TArray {OF PACKED ARRAY [1..SIZEOF(TClassName)] OF CHAR};
000185          END;
000186
000187      TEnumAccesses = (fRead, fWrite, fAppend, fPrivate); {not allowing global_refnum at this time}
```

```
000188      TAccesses = SET OF TEnumAccesses;
000189      TIOMode = (fAbsolute, fRelative, fSequential);
000190      xReadWrite = (xRead, xWrite);
000191      SizeOfNumber = 1..4;
000192
000193      TScanDirection = (scanForward, scanBackward);
000194
000195      TConvResult = (cvValid, cvNoNumber, cvBadNumber, cvOverflow);
000196
000197
000198    {Classes}
000199
000200      TObject = SUBCLASS OF NIL
000201
000202        {Creation and Destruction}
000203          FUNCTION  TObject.CREATE(object: TObject; heap: THeap): TObject; ABSTRACT;
000204          PROCEDURE TObject.Become(object: TObject);              {SELF becomes obj and former SELF is freed}
000205          FUNCTION  TObject.Class: TClass;                        {its class pointer}
000206          FUNCTION  TObject.CloneObject(heap: THeap): TObject;    {clones just the object, not its dependents}
000207          FUNCTION  TObject.Clone(heap: THeap): TObject; DEFAULT; {clones the object and its known dependents}
000208          PROCEDURE TObject.FreeObject; DEFAULT;                  {frees just the object, not its dependents}
000209          PROCEDURE TObject.Free; DEFAULT;                        {frees the object and its known dependents}
000210          FUNCTION  TObject.Heap: THeap;                          {which heap it is in}
000211          FUNCTION  TObject.HeapBytes: INTEGER;                   {number of bytes occupied in that heap}
000212          PROCEDURE TObject.Read(s: TStringScanner);              {reads the object & its known dependents}
000213          PROCEDURE TObject.Write(s: TStringScanner);             {writes the object & its known dependents}
000214
000215        {Debugging}
000216          {$IFC fDebugMethods}
000217          PROCEDURE TObject.Fields(PROCEDURE Field(nameAndType: S255)); DEFAULT; {See end of file for comment}
000218          PROCEDURE TObject.Debug(numLevels: INTEGER; memberTypeStr: S255); DEFAULT;
000219           {writes an object down to numLevels:
000220                    numLevels=0 => write only class;
000221                    numLevels=1 => write class, non-Object fields, and class of Object fields
000222                    etc.}
000223          {$ENDC}
000224
000225        {Version Conversion}
000226          PROCEDURE TObject.Convert(fromVersion: Byte);  {Override it to finish conversion from an old version}
000227          FUNCTION  TObject.JoinClass(newClass: TClass): TObject;       {Called for you by version conversion}
000228
000229          END;
000230
000231
000232      TCollecHeader =  RECORD
000233          classPtr:       TClass;
000234          size:           LONGINT;    {number of real elements, not counting the hole}
000235          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data; MAXINT if none allowed}
```

```
000236          holeStart:      INTEGER;    {0 = at the beginning, size = at the end; MAXINT = none allowed}
000237          holeSize:       INTEGER;    {measured in MemberBytes units}
000238          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grow the collection by}
000239          END;
000240
000241      TFastString = RECORD            {only access ch[i] when hole is at end & TString is not subclassed}
000242          header:         TCollecHeader;
000243          ch:             PACKED ARRAY[1..32740] OF CHAR;
000244          END;
000245      TPFastString = ^TFastString;
000246      THFastString = ^TPFastString;
000247
000248      TArrayHeader =  RECORD
000249          classPtr:       TClass;
000250          size:           LONGINT;    {number of real elements, not counting the hole}
000251          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data}
000252          holeStart:      INTEGER;    {0 means hole at the beginning, size means hole at the end}
000253          holeSize:       INTEGER;    {measured in MemberBytes units}
000254          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grow the collection by}
000255          recordBytes:    INTEGER;
000256          END;
000257
000258
000259      TCollection = SUBCLASS OF TObject
000260
000261        {Variables}
000262          size:           LONGINT;    {number of real elements, not counting the hole}
000263          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data}
000264          holeStart:      INTEGER;    {0 means hole at the beginning, size means hole at the end}
000265          holeSize:       INTEGER;    {measured in MemberBytes units}
000266          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grow the collection by}
000267
000268          {The field "size" is a LONGINT for the benefit of huge collections like remote data bases.
000269           It is always in the INTEGER range for non-subclassed TLists, TArrays, and TStrings.
000270
000271           The field "dynStart" is an offset from Handle(collection)^ and tells where the dynamic part
000272           of the data is stored, if any.  This convention allows subclasses to add fields.
000273
000274           When editing a collection, there may be an unused "hole" somewhere in the storage block.  The
000275           fields "holeStart" and "holeSize" specify (in member-sized units) the starting index of the
000276           hole and the length of the hole.  When holeSize is zero, there is no hole.  If members are
000277           added when there is no hole, the storage block is expanded to allow for at least another
000278           "holeStd" members.
000279
000280           CREATE has an argument that lets the initial collection have a hole at the end, so that
000281           Ins- methods can be called to initialize the collection without any storage allocation.
000282
000283           StartEdit sets holeStd to its argument, which forces subsequent edit methods to leave intact
```

```
000284          any hole they might form. StopEdit squeezes out the hole and sets holeStd to zero, which
000285          forces subsequent edit methods that get called with no hole to squeeze out any hole they may form.
000286          Thus, every StartEdit that has a nonzero argument should be terminated by a call on StopEdit to
000287          save space.}
000288
000289        {Creation and Destruction}
000290          FUNCTION  TCollection.CREATE(object: TObject; heap: THeap; initialSlack: INTEGER): TCollection;
000291          FUNCTION  TCollection.Clone(heap: THeap): TObject; OVERRIDE;
000292
000293        {Attributes}
000294          FUNCTION  TCollection.MemberBytes: INTEGER; ABSTRACT;
000295          FUNCTION  TCollection.Equals(otherCollection: TCollection): BOOLEAN;
000296
000297        {Slack control}
000298          PROCEDURE TCollection.StartEdit(withSlack: INTEGER);
000299          PROCEDURE TCollection.StopEdit;
000300
000301        {Generic Inserts}
000302          PROCEDURE TCollection.InsManyAt(i: LONGINT; otherCollection: TCollection; index, howMany: LONGINT);
000303          PROCEDURE TCollection.InsNullsAt(i, howMany: LONGINT);
000304
000305      (* BEGIN CONCEPTUAL METHODS (parameter types differ in subclasses; sometimes extra parameters required)
000306
000307        {Enumerate members}
000308          PROCEDURE TCollection.Each(PROCEDURE DoToMember(member: "TMember")); CONCEPTUAL;
000309          FUNCTION  TCollection.Pos(after: LONGINT; member: "TMember"): LONGINT; CONCEPTUAL;
000310          FUNCTION  TCollection.Scanner: TScanner; CONCEPTUAL;           {c.ScannerFrom(-MaxLInt, scanForward)}
000311          FUNCTION  TCollection.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection)
000312                                          : TScanner; CONCEPTUAL;
000313
000314        {Inspect members}
000315          FUNCTION  TCollection.At(i: LONGINT): "TMember"; CONCEPTUAL;
000316          FUNCTION  TCollection.First: "TMember"; CONCEPTUAL;
000317          FUNCTION  TCollection.Last: "TMember"; CONCEPTUAL;
000318          FUNCTION  TCollection.ManyAt(i, howMany: LONGINT): "TCollection"; CONCEPTUAL;
000319
000320        {Insert members}
000321          PROCEDURE TCollection.InsAt(i: LONGINT; member: "TMember"); CONCEPTUAL;
000322          PROCEDURE TCollection.InsFirst(member: "TMember"); CONCEPTUAL;
000323          PROCEDURE TCollection.InsLast(member: "TMember"); CONCEPTUAL;
000324
000325        {Delete members}
000326          PROCEDURE TCollection.DelAll; CONCEPTUAL;
000327          PROCEDURE TCollection.DelAt(i: LONGINT); CONCEPTUAL;
000328          PROCEDURE TCollection.DelFirst; CONCEPTUAL;
000329          PROCEDURE TCollection.DelLast; CONCEPTUAL;
000330          PROCEDURE TCollection.DelManyAt(i, howMany: LONGINT); CONCEPTUAL;
000331
```

```
000332        {Change member}
000333          PROCEDURE TCollection.PutAt(i: LONGINT; member: "TMember"); CONCEPTUAL;
000334
000335      END CONCEPTUAL METHODS *)
000336
000337        {Private methods -- to be called by subclasses only!!!}
000338          {$IFC fRngObject}
000339          PROCEDURE TCollection.CheckIndex(index: LONGINT);
000340          {$ENDC}
000341          FUNCTION  TCollection.AddrMember(i: LONGINT): LONGINT;       {The address is only valid momentarily}
000342          PROCEDURE TCollection.CopyMembers(dstAddr, startIndex, howMany: LONGINT);
000343          PROCEDURE TCollection.EditAt(atIndex: LONGINT; deltaMembers: INTEGER);           {Transfers no data}
000344          PROCEDURE TCollection.ResizeColl(membersPlusHole: INTEGER);      {Resizes at end, no fields changed}
000345          PROCEDURE TCollection.ShiftColl(afterSrcIndex, afterDstIndex, howMany: INTEGER); {No fields changed}
000346
000347          END;
000348
000349       TList = SUBCLASS OF TCollection
000350
000351        {Variables}
000352
000353        {Creation and Destruction}
000354          FUNCTION  TList.CREATE(object: TObject; heap: THeap; initialSlack: INTEGER): TList;
000355          FUNCTION  TList.Clone(heap: THeap): TObject; OVERRIDE;
000356          PROCEDURE TList.Free; OVERRIDE;
000357
000358        {Debugging}
000359         {$IFC fDebugMethods}
000360          PROCEDURE TList.Debug(numLevels: INTEGER; memberTypeStr: S255); OVERRIDE;
000361              { numLevels=0   print just class of list;
000362                          1   also print size of list;
000363                          2   also print compacted list of member classes
000364                       >=3   print class, size, and call Debug(numLevels-1) on members
000365              }
000366          PROCEDURE TList.DebugMembers;
000367         {$ENDC}
000368
000369        {Attributes}
000370          FUNCTION  TList.MemberBytes: INTEGER; OVERRIDE;
000371
000372        {Enumerate members}
000373          PROCEDURE TList.Each(PROCEDURE DoToObject(object: TObject)); DEFAULT;
000374          FUNCTION  TList.Pos(after: LONGINT; object: TObject): LONGINT;
000375          FUNCTION  TList.Scanner: TListScanner;
000376          FUNCTION  TList.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection)
000377                                    : TListScanner; DEFAULT;
000378
000379        {Inspect members}
```

```
000380          FUNCTION  TList.At(i: LONGINT): TObject; DEFAULT;
000381          FUNCTION  TList.First: TObject; DEFAULT;
000382          FUNCTION  TList.Last: TObject; DEFAULT;
000383          FUNCTION  TList.ManyAt(i, howMany: LONGINT): TList; DEFAULT;
000384
000385        {Insert members}
000386          PROCEDURE TList.InsAt(i: LONGINT; object: TObject); DEFAULT;
000387          PROCEDURE TList.InsFirst(object: TObject);
000388          PROCEDURE TList.InsLast(object: TObject);
000389
000390        {Delete members}
000391          PROCEDURE TList.DelAll(freeOld: BOOLEAN); DEFAULT;
000392          PROCEDURE TList.DelAt(i: LONGINT; freeOld: BOOLEAN); DEFAULT;
000393          PROCEDURE TList.DelFirst(freeOld: BOOLEAN);
000394          PROCEDURE TList.DelLast(freeOld: BOOLEAN);
000395          PROCEDURE TList.DelManyAt(i, howMany: LONGINT; freeOld: BOOLEAN); DEFAULT;
000396          PROCEDURE TList.DelObject(object: TObject; freeOld: BOOLEAN);
000397          FUNCTION  TList.PopLast: TObject;
000398
000399        {Change member}
000400          PROCEDURE TList.PutAt(i: LONGINT; object: TObject; freeOld: BOOLEAN); DEFAULT;
000401
000402          END;
000403
000404      TArray = SUBCLASS OF TCollection     {*** WARNING: The Ptrs below become invalid if the heap compacts!!!}
000405
000406        {Variables}
000407          recordBytes: INTEGER;
000408
000409        {Creation and Destruction}
000410          FUNCTION  TArray.CREATE(object: TObject; heap: THeap; initialSlack, bytesPerRecord: INTEGER): TArray;
000411
000412        {Attributes}
000413          FUNCTION  TArray.MemberBytes: INTEGER; OVERRIDE;
000414
000415        {Enumerate members}
000416          PROCEDURE TArray.Each(PROCEDURE DoToRecord(pRecord: Ptr)); DEFAULT;
000417          FUNCTION  TArray.Pos(after: LONGINT; pRecord: Ptr): LONGINT;
000418          FUNCTION  TArray.Scanner: TArrayScanner;
000419          FUNCTION  TArray.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection)
000420                                       : TArrayScanner; DEFAULT;
000421
000422        {Inspect members}
000423          FUNCTION  TArray.At(i: LONGINT): Ptr; DEFAULT;
000424          FUNCTION  TArray.First: Ptr;
000425          PROCEDURE TArray.GetAt(i: LONGINT; pRecord: Ptr); DEFAULT;  {Sort of: pRecord^ := SELF.At(i)^}
000426          FUNCTION  TArray.Last: Ptr;
000427          FUNCTION  TArray.ManyAt(i, howMany: LONGINT): TArray; DEFAULT;
```

```
000428
000429        {Insert members}
000430          PROCEDURE TArray.InsAt(i: LONGINT; pRecord: Ptr); DEFAULT;
000431          PROCEDURE TArray.InsFirst(pRecord: Ptr);
000432          PROCEDURE TArray.InsLast(pRecord: Ptr);
000433
000434        {Delete members}
000435          PROCEDURE TArray.DelAll; DEFAULT;
000436          PROCEDURE TArray.DelAt(i: LONGINT); DEFAULT;
000437          PROCEDURE TArray.DelFirst;
000438          PROCEDURE TArray.DelLast;
000439          PROCEDURE TArray.DelManyAt(i, howMany: LONGINT); DEFAULT;
000440
000441        {Change member}
000442          PROCEDURE TArray.PutAt(i: LONGINT; pRecord: Ptr); DEFAULT;
000443
000444          END;
000445
000446      TString = SUBCLASS OF TCollection
000447
000448        {Variables}
000449
000450        {Creation and Destruction}
000451          FUNCTION  TString.CREATE(object: TObject; heap: THeap; initialSlack: INTEGER): TString;
000452
000453        {Attributes}
000454          FUNCTION  TString.MemberBytes: INTEGER; OVERRIDE;
000455
000456        {Enumerate members}
000457          PROCEDURE TString.Each(PROCEDURE DoToCharacter(character: CHAR));
000458          FUNCTION  TString.Pos(after: LONGINT; character: CHAR): LONGINT;
000459          FUNCTION  TString.Scanner: TStringScanner;
000460          FUNCTION  TString.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection): TStringScanner;
000461
000462        {Inspect members}
000463          FUNCTION  TString.At(i: LONGINT): CHAR;
000464          FUNCTION  TString.First: CHAR;
000465          FUNCTION  TString.Last: CHAR;
000466          FUNCTION  TString.ManyAt(i, howMany: LONGINT): TString;
000467          PROCEDURE TString.ToPStr(pStr: TPString);
000468          PROCEDURE TString.ToPStrAt(i, howMany: LONGINT; pStr: TPString);
000469
000470        {Insert members}
000471          PROCEDURE TString.InsAt(i: LONGINT; character: CHAR);
000472          PROCEDURE TString.InsFirst(character: CHAR);
000473          PROCEDURE TString.InsLast(character: CHAR);
000474          PROCEDURE TString.InsPStrAt(i: LONGINT; pStr: TPString);
000475
```

```
000476        {Delete members}
000477          PROCEDURE TString.DelAll;
000478          PROCEDURE TString.DelAt(i: LONGINT);
000479          PROCEDURE TString.DelFirst;
000480          PROCEDURE TString.DelLast;
000481          PROCEDURE TString.DelManyAt(i, howMany: LONGINT);
000482
000483        {Change member}
000484          PROCEDURE TString.PutAt(i: LONGINT; character: CHAR);
000485
000486        {QuickDraw}
000487          PROCEDURE TString.Draw(i: LONGINT; howMany: INTEGER);
000488          FUNCTION  TString.Width(i: LONGINT; howMany: INTEGER): INTEGER;
000489
000490          END;
000491
000492      TFile = SUBCLASS OF TCollection
000493
000494        {Variables}
000495          path:       TFilePath;
000496          password:   TPassword;              {The current password protecting this file, and used for all
000497                                                  accesses to it;  client is responsible for setting this
000498                                                  field after the TFile is created; ignored if
000499                                                  LibraryVersion <= 20}
000500          scanners:   TList {OF TScanner};
000501
000502        {Creation and Destruction}
000503          FUNCTION  TFile.CREATE(object: TObject; heap: THeap; itsPath: TFilePath;
000504                                 itsPassword: TPassword): TFile;
000505                    {itsPassword is ignored from LibraryVersion <= 20}
000506
000507          PROCEDURE TFile.Free; OVERRIDE;                          {Frees the scanners as well}
000508          FUNCTION  TFile.Clone(heap: THeap): TObject; OVERRIDE;  {Illegal}
000509
000510        {Attributes}
000511          FUNCTION  TFile.MemberBytes: INTEGER; OVERRIDE;
000512
000513        {Enumerate members}
000514          FUNCTION  TFile.Scanner: TFileScanner;                   {f.ScannerFrom(0, [fRead, fWrite])}
000515          FUNCTION  TFile.ScannerFrom(firstToScan: LONGINT; manip: TAccesses): TFileScanner;
000516
000517        {Catalog}
000518          PROCEDURE TFile.ChangePassword(VAR error: INTEGER; newPassword: TPassword);
000519                      {also changes the password field, if successful}
000520          PROCEDURE TFile.Delete(VAR error: INTEGER);
000521          FUNCTION  TFile.Exists(VAR error: INTEGER): BOOLEAN;
000522          FUNCTION  TFile.WhenModified(VAR error: INTEGER): LONGINT;
000523          PROCEDURE TFile.Rename(VAR error: INTEGER; newFileName: TFilePath);
```

```
000524          FUNCTION  TFile.VerifyPassword(VAR error: INTEGER; password: TPassword): BOOLEAN;
000525
000526          END;
000527
000528       TScanner = SUBCLASS OF TObject
000529
000530         {Variables}
000531           collection:     TCollection; {The collection being scanned}
000532           position:       LONGINT;     {The current position (between members: 0=before first, size+1=after
000533                                         last)}
000534           increment:      INTEGER;     {1 if scanning forward, -1 if scanning backward}
000535           scanDone:       BOOLEAN;     {TRUE if next .Scan call should return FALSE, leaving its VAR
000536                                         parameter alone}
000537           atEnd:          BOOLEAN;     {TRUE if next .Scan call will return FALSE because at end of collection}
000538
000539           FUNCTION  TScanner.CREATE(object: TObject; itsCollection: TCollection; itsInitialPosition: LONGINT;
000540                                     scanDirection: TScanDirection): TScanner;
000541
000542         {Close and Reopen}
000543           PROCEDURE TScanner.Close; DEFAULT;  {If disk-based, flush buffers and tell OS to close file,
000544                                                else no-op}
000545           PROCEDURE TScanner.Open; DEFAULT;   {If disk-based, tell OS to reopen file and fill first buffer}
000546
000547         {Slack Control}
000548           PROCEDURE TScanner.Allocate(slack: LONGINT); DEFAULT;   {Like collection.StartEdit(slack)}
000549           PROCEDURE TScanner.Compact; DEFAULT;                    {Like collection.StopEdit}
000550
000551         {Positioning}
000552           FUNCTION  TScanner.Advance(PROCEDURE DoToCurrent(anotherMember: BOOLEAN)): BOOLEAN;
000553           PROCEDURE TScanner.Done; DEFAULT;                       {Set scanDone so that Scan will return FALSE}
000554           PROCEDURE TScanner.Reverse; DEFAULT;                    {Reverse the scan direction}
000555           PROCEDURE TScanner.Seek(newPosition: LONGINT); DEFAULT; {Forces to legal places}
000556           PROCEDURE TScanner.Skip(deltaPos: LONGINT); DEFAULT;    {Forces to legal places}
000557
000558       (* BEGIN CONCEPTUAL METHODS (parameter types differ in subclasses; sometimes extra parameters required)
000559
000560         {Data Transfer}
000561           FUNCTION  TScanner.Obtain: "TMember"; CONCEPTUAL; {Return previous member  (redundant right after
000562                                                              Scan)}
000563           FUNCTION  TScanner.Scan(VAR member: "TMember"): BOOLEAN; CONCEPTUAL; {Return next & advance past it}
000564
000565         {Editing}
000566           PROCEDURE TScanner.Append(member: "TMember"); CONCEPTUAL;   {Add a new member after position, scan
000567                                                                       past it}
000568           PROCEDURE TScanner.Delete; CONCEPTUAL;                     {Delete previous member and adjust
000569                                                                       position}
000570           PROCEDURE TScanner.DeleteRest; CONCEPTUAL;                 {Delete everything after SELF.position}
000571           PROCEDURE TScanner.Replace(member: "TMember"); CONCEPTUAL;  {Replace previous member and maintain
```

```
000572                                                                  position}
000573
000574     END CONCEPTUAL METHODS *)
000575
000576          END;
000577
000578      TListScanner = SUBCLASS OF TScanner
000579
000580        {Variables}
000581
000582        {Creation and Destruction}
000583          FUNCTION  TListScanner.CREATE(object: TObject; itsList: TList; itsInitialPosition: LONGINT;
000584                                        itsScanDirection: TScanDirection): TListScanner;
000585          PROCEDURE TListScanner.Free; OVERRIDE;
000586
000587        {Traversal}
000588          FUNCTION  TListScanner.Obtain: TObject; DEFAULT; {Return previous member (redundant right after Scan)}
000589          FUNCTION  TListScanner.Scan(VAR nextObject: TObject): BOOLEAN; DEFAULT;{Return next, advance past it}
000590
000591        {Editing}
000592          PROCEDURE TListScanner.Append(object: TObject); DEFAULT;   {Add object after position, scan past it}
000593          PROCEDURE TListScanner.Delete(freeOld: BOOLEAN); DEFAULT;  {Delete previous object, adjust position}
000594          PROCEDURE TListScanner.DeleteRest(freeOld: BOOLEAN); DEFAULT; {Delete all objects after position}
000595          PROCEDURE TListScanner.Replace(object: TObject; freeOld: BOOLEAN); DEFAULT;     {Replace previous}
000596
000597          END;
000598
000599      TArrayScanner = SUBCLASS OF TScanner
000600
000601        {Variables}
000602
000603        {Creation and Destruction}
000604          FUNCTION  TArrayScanner.CREATE(object: TObject; itsArray: TArray; itsInitialPosition: LONGINT;
000605                                         itsScanDirection: TScanDirection): TArrayScanner;
000606          PROCEDURE TArrayScanner.Free; OVERRIDE;
000607
000608        {Traversal}
000609          FUNCTION  TArrayScanner.Obtain: Ptr; DEFAULT; {Return previous member (redundant right after Scan)}
000610          FUNCTION  TArrayScanner.Scan(VAR pNextRecord: Ptr): BOOLEAN; DEFAULT; {Return next & advance past it}
000611
000612        {Editing}
000613          PROCEDURE TArrayScanner.Append(pRecord: Ptr); DEFAULT; {Add a new record after position, scan past it}
000614          PROCEDURE TArrayScanner.Delete; DEFAULT;               {Delete previous record and adjust position}
000615          PROCEDURE TArrayScanner.DeleteRest; DEFAULT;           {Delete all records after position}
000616          PROCEDURE TArrayScanner.Replace(pRecord: Ptr); DEFAULT;{Replace previous record and maintain position}
000617
000618          END;
000619
```

```
000620      TStringScanner = SUBCLASS OF TScanner
000621
000622        {Variables}
000623          actual: LONGINT;                                {no. bytes last xfered}
000624
000625        {Creation and Destruction}
000626          FUNCTION  TStringScanner.CREATE(object: TObject; itsString: TString; itsInitialPosition: LONGINT;
000627                                          itsScanDirection: TScanDirection): TStringScanner;
000628          PROCEDURE TStringScanner.Free; OVERRIDE;
000629
000630        {Traversal}
000631          FUNCTION  TStringScanner.Obtain: CHAR; DEFAULT; {Return previous member (redundant right after Scan)}
000632          FUNCTION  TStringScanner.Scan(VAR nextChar: CHAR): BOOLEAN; DEFAULT;   {Return next & advance past it}
000633
000634        {Editing}
000635          PROCEDURE TStringScanner.Append(character: CHAR); DEFAULT; {Add char after position, scan past it}
000636          PROCEDURE TStringScanner.Delete; DEFAULT;                  {Delete previous char, adjust position}
000637          PROCEDURE TStringScanner.DeleteRest; DEFAULT;              {Delete all chars after position}
000638          PROCEDURE TStringScanner.Replace(character: CHAR); DEFAULT;{Replace previous char, maintain position}
000639
000640        {Typed Sequential Data Transfer: characters are read/written from left to right regardless of increment}
000641          FUNCTION  TStringScanner.ReadArray(heap: THeap; bytesPerRecord: INTEGER): TArray; {reads size first}
000642          FUNCTION  TStringScanner.ReadNumber(numBytes: SizeOfNumber): LONGINT;  {iff numBytes is even
000643                                                                                  then signed}
000644          FUNCTION  TStringScanner.ReadObject(heap: THeap): TObject;   {tells object to Read(SELF)}
000645          PROCEDURE TStringScanner.WriteArray(a: TArray);  {inverse of ReadArray: writes size but not
000646                                                            recordBytes}
000647          PROCEDURE TStringScanner.WriteNumber(value: LONGINT; numBytes: SizeOfNumber);{does not write size}
000648          PROCEDURE TStringScanner.WriteObject(object: TObject);  {tells object to Write(SELF)}
000649          PROCEDURE TStringScanner.XferContiguous(whichWay: xReadWrite; collection: TCollection);
000650                           {xfers the size and members, non-recursively; xRead appends what it reads}
000651          PROCEDURE TStringScanner.XferFields(whichWay: xReadWrite; object: TObject); {xfers all but the class}
000652          PROCEDURE TStringScanner.XferPString(whichWay: xReadWrite; pStr: TPString); {it better be long enough}
000653
000654        {Untyped Data Transfer: characters are read/written from left to right regardless of increment}
000655          PROCEDURE TStringScanner.XferSequential(whichWay: xReadWrite; pFirst: Ptr; numBytes:
000656                                                  LONGINT); DEFAULT;
000657          PROCEDURE TStringScanner.XferRandom(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT;
000658                                              mode: TIOMode; offset: LONGINT); DEFAULT;
000659
000660          END;
000661
000662      TFileScanner = SUBCLASS OF TStringScanner
000663
000664        {Variables}
000665          accesses:   TAccesses;                          {[fRead, fWrite, fAppend, fPrivate]}
000666          refnum:     INTEGER;                            {OS file refnum, or -1 if not open now}
000667          error:      INTEGER;    {EOF is not an error}   {first error (or warning if no error) encountered}
```

```
000668
000669        {Creation and Destruction}
000670          FUNCTION  TFileScanner.CREATE(object: TObject; itsFile: TFile; manip: TAccesses): TFileScanner;
000671          PROCEDURE TFileScanner.FreeObject; OVERRIDE;        {also closes the OS file}
000672          PROCEDURE TFileScanner.Free; OVERRIDE;              {if the last scanner, frees the TFile, too}
000673
000674        {Close and Reopen}
000675          PROCEDURE TFileScanner.Close; OVERRIDE;
000676          PROCEDURE TFileScanner.Open; OVERRIDE;
000677
000678        {Slack Control}
000679          PROCEDURE TFileScanner.Allocate(slack: LONGINT); OVERRIDE; {Get slack DIV pageSize unused disk pages}
000680          PROCEDURE TFileScanner.Compact; OVERRIDE;                  {Return unused disk pages to free space}
000681
000682        {Positioning}
000683          PROCEDURE TFileScanner.Seek(newPosition: LONGINT); OVERRIDE;
000684          PROCEDURE TFileScanner.Skip(deltaPos: LONGINT); OVERRIDE;
000685
000686        {Traversal}
000687          FUNCTION  TFileScanner.Obtain: CHAR; OVERRIDE; {Return previous member (redundant right after Scan)}
000688          FUNCTION  TFileScanner.Scan(VAR nextChar: CHAR): BOOLEAN; OVERRIDE;   {Return next & advance past it}
000689
000690        {Editing}
000691          PROCEDURE TFileScanner.Append(character: CHAR); OVERRIDE; {Acts like: Replace; Skip(1)}
000692          PROCEDURE TFileScanner.Delete; OVERRIDE;                  {Acts like: Skip(-1)}
000693          PROCEDURE TFileScanner.DeleteRest; OVERRIDE;              {Shorten file size to SELF.position}
000694          PROCEDURE TFileScanner.Replace(character: CHAR); OVERRIDE;{Replace previous member and maintain
000695                                                                     position}
000696
000697        {Untyped Data Transfer: characters are read/written from left to right regardless of increment}
000698          PROCEDURE TFileScanner.XferSequential(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT); OVERRIDE;
000699          PROCEDURE TFileScanner.XferRandom(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT;
000700                                            mode: TIOMode; offset: LONGINT); OVERRIDE;
000701
000702         END;
000703
000704    {$IFC compatibleLists}
000705
000706        {Backward compatibility classes}
000707
000708        TDynamicArray = SUBCLASS OF TArray
000709            ch: {UNPACKED} ARRAY [0..16370] OF CHAR;
000710            FUNCTION  TDynamicArray.CREATE(object: TObject; heap: THeap; bytesPerRecord: INTEGER;
000711                                           initialSize: INTEGER): TDynamicArray;
000712            FUNCTION  TDynamicArray.NumRecords: INTEGER;
000713            PROCEDURE TDynamicArray.BeSize(newSize: INTEGER);
000714            END;
000715
```

```
000716      TIndexList = SUBCLASS OF TList
000717          elements: ARRAY[1..1] OF TObject;
000718          FUNCTION  TIndexList.CREATE(object: TObject; heap: THeap; initialSize: INTEGER): TIndexList;
000719          FUNCTION  TIndexList.numElements: INTEGER;
000720          END;
000721
000722      TLinkList = SUBCLASS OF TList
000723          FUNCTION  TLinkList.CREATE(object: TObject; heap: THeap): TLinkList;
000724          FUNCTION  TLinkList.numElements: INTEGER;
000725          END;
000726
000727      TBlockList = SUBCLASS OF TList
000728          FUNCTION  TBlockList.CREATE(object: TObject; heap: THeap; itsMinBlockSize: INTEGER): TBlockList;
000729          FUNCTION  TBlockList.numElements: INTEGER;
000730          END;
000731
000732      TFileStream = SUBCLASS OF TFileScanner
000733          FUNCTION  TFileStream.CREATE(object: TObject; heap: THeap; path: S255; manip: TAccesses): TFileStream;
000734          FUNCTION  TFileStream.Size: LONGINT;
000735          END;
000736
000737  {$ENDC}
000738
000739
000740  VAR
000741
000742      mainDsRefnum:           INTEGER;                    {refnum of the process data segment}
000743      mainHeap:               THeap;                      {heap of the process}
000744      mainLdsn:               INTEGER;                    {ldsn of the process data segment}
000745      fCheckIndices:          BOOLEAN;
000746
000747      onDesktop:              BOOLEAN;                    {Is there a DM (Desktop Manager) to talk to?}
000748      wmIsInitialized:        BOOLEAN;                    {Has OpenWM been done?}
000749      isInitialized:          BOOLEAN;                    {Iff TRUE, shouldn't tell DM initFailed any more}
000750      amDying:                BOOLEAN;                    {Iff TRUE, I have called ImDying}
000751
000752      myWorld:                TClassWorld;                {For Version Conversion}
000753
000754
000755      { Variables for Debugging }
000756          indentTrace:    INTEGER;
000757
000758            { stuff for the intelligent output }
000759          currXPos:       INTEGER;
000760          outputIndent:   INTEGER;
000761
000762          {$IFC fTrace}
000763              { TRUE if we want to inhibit tracing; client must save and restore its value;
```

```
000764                      normally this is needed only if you override the Debug method }
000765          fDebugRecursion:    BOOLEAN;
000766              { how often to call KeyPress from debugger to check for user interrupt }
000767          keyPresLimit:       INTEGER;
000768          {$ENDC}
000769
000770
000771  {$IFC fCheckHeap}
000772  FUNCTION CountHeap(heap: THeap): INTEGER;
000773  {$ENDC}
000774
000775  FUNCTION  Min(i, j: LONGINT): LONGINT;
000776  FUNCTION  Max(i, j: LONGINT): LONGINT;
000777
000778  PROCEDURE XferLeft(source, dest: Ptr; nBytes: INTEGER);
000779  PROCEDURE XferRight(source, dest: Ptr; nBytes: INTEGER);
000780  FUNCTION  EqualBytes(source, dest: Ptr; nBytes: INTEGER): BOOLEAN;
000781
000782  FUNCTION  LIntAndLInt(i, j: LONGINT): LONGINT;
000783  FUNCTION  LIntOrLInt(i, j: LONGINT): LONGINT;
000784  FUNCTION  LIntXorLInt(i, j: LONGINT): LONGINT;
000785
000786  FUNCTION  NewObject(heap: THeap; itsClass: TClass): TObject;
000787  FUNCTION  NewDynObject(heap: THeap; itsClass: TClass; dynBytes: INTEGER): TObject;
000788  PROCEDURE ResizeDynObject(object: TObject; newTotalBytes: INTEGER);
000789  FUNCTION  NewOrRecycledObject(heap: THeap; itsClass: TClass; VAR chainHead: TObject): TObject;
000790  PROCEDURE RecycleObject(object: TObject; VAR chainHead: TObject);
000791  PROCEDURE Free(object: TObject);
000792
000793  {$IFC compatibleLists}
000794      {Backward compatibility procedures}
000795
000796  FUNCTION  SubObject(super: TObject; itsClass: TClass): TObject;
000797  PROCEDURE FileDelete(path: S255);
000798  PROCEDURE FileLookup(VAR error: INTEGER; path: S255);
000799  PROCEDURE FileRename(oldPath, newPath: S255);
000800  FUNCTION  FileModified(path: S255): LONGINT;
000801  {$ENDC}
000802
000803  FUNCTION  Superclass(class: TClass): TClass;
000804  FUNCTION  ClassDescendsFrom(descendant, ancestor: TClass): BOOLEAN;
000805
000806  PROCEDURE NameOfClass(class: TClass; VAR className: TClassName);
000807  FUNCTION  SizeOfClass(class: TClass): INTEGER;
000808
000809  {The next 3 can only be called from a class-init block or a subroutine of a class-init block}
000810  PROCEDURE UnitAuthor(companyAndAuthor: TAuthorName);                          {required once per unit}
000811  PROCEDURE ClassAuthor(companyAndAuthor: TAuthorName; classAlias: TClassName);   {optional}
```

```
000812  PROCEDURE ClassVersion(itsVersion, oldestItCanRead: Byte);                    {optional}
000813
000814  FUNCTION  ValidObject(hndl: Handle): BOOLEAN;
000815
000816  PROCEDURE ABCBreak(s: S255; errCode: LONGINT);
000817
000818  PROCEDURE ClascalError(error: INTEGER);
000819
000820  {Some useful procedures; we should decide once and for all whether or not to keep any or all of these}
000821  PROCEDURE LIntToHex(decNumber: LONGINT; hexNumber: TPString);
000822              {NOTE: hexNumber must be >= 8 characters, regardless of size of decNumber}
000823  PROCEDURE LIntToStr(decNumber: LONGINT; str: TPString);
000824              {NOTE: str must be >= 11 characters (sign + 10 digits), regardless of size of decNumber}
000825  PROCEDURE IntToStr(decNumber: INTEGER; str: TPString);
000826              {NOTE: str must be >= 6 characters (sign + 5 digits), regardless of size of decNumber}
000827  PROCEDURE HexStrToLInt(hexString: TPString; VAR decNumber: LONGINT; VAR result: TConvResult);
000828  PROCEDURE StrToLInt(str: TPString; VAR decNumber: LONGINT; VAR result: TConvResult);
000829  PROCEDURE StrToInt(str: TPString; VAR decNumber: INTEGER; VAR result: TConvResult);
000830
000831  PROCEDURE TrimBlanks(str: TPString);
000832  FUNCTION  CharUpperCased(ch: CHAR): CHAR;
000833  PROCEDURE StrUpperCased(str: TPString);
000834
000835  PROCEDURE SplitFilePath(VAR fullPath, itsCatalog, itsFilePart: TFilePath);
000836      {fullPath = CONCAT(itsCatalog, itsFilePart}
000837
000838  PROCEDURE LatestError(newError: INTEGER; VAR previousError: INTEGER);
000839          {This is used to handle error codes returned by multiple operations, so that you end up with
000840              the first error number or warning number (error code < 0) if there was no error.
000841           You should pass in the latest error as 'newError' and the variable that is to be the final
000842              error code as 'previousError'.  Here is the actual code of LatestError:
000843
000844                              IF ((newError > 0) AND (previousError <= 0) OR
000845                                  (newError < 0) AND (previousError = 0)) THEN
000846                                 previousError := newError
000847          }
000848
000849  {$IFC fDbgObject}
000850  PROCEDURE EntDebugger(inputStr, enterReason: S255);
000851  PROCEDURE DumpVar(pVariable: Ptr; nameAndType: S255); {used mainly by TProcess.DumpGlobals}
000852  PROCEDURE WrStr(str: S255);         { write a string with wrap-around }
000853  PROCEDURE WrLn;                     { goto next line, and output indent }
000854  {$IFC fDebugMethods}
000855  PROCEDURE WrObj(object: TObject; numLevels: INTEGER; memberTypeStr: S255);
000856  {$ENDC}
000857  {$ENDC}
000858
000859  {$IFC fDbgObject OR fDebugMethods}
```

```
000860  FUNCTION CheckKeyPress(routine: S255): BOOLEAN;
000861  {$ENDC}
000862
000863  FUNCTION  NewHeap(VAR error: INTEGER; heapStart, numBytes: LONGINT; numObjects: INTEGER): THeap;
000864  FUNCTION  MakeDataSegment(VAR error, dsRefnum: INTEGER; firstTryVolume, thenTryVolume: TFilePath;
000865                            ldsn, memBytes, diskBytes: INTEGER): LONGINT;
000866
000867  PROCEDURE SetHeap(heap: THeap);
000868  PROCEDURE GetHeap(VAR heap: THeap);
000869      {We can't USE Unit Storage because of type name conflicts (Ptr, Handle, ProcPtr)}
000870
000871
000872  FUNCTION  NeedConversion(exClassWorld: TClassWorld; VAR olderVersion, newerVersion: BOOLEAN): BOOLEAN;
000873  PROCEDURE ConvertHeap(heap: THeap; exClassWorld: TClassWorld);
000874
000875  PROCEDURE MarkHeap(heap: THeap; mpAddress: LONGINT);
000876  PROCEDURE SweepHeap(heap: THeap; report: BOOLEAN);
000877
000878  {$IFC fTrace}
000879  PROCEDURE BP(MyTraceLevel:integer);
000880                {Trace entry to method and write SELF (unless CREATE, Debug, or FreeObject)}
000881  PROCEDURE EP;  {Trace entry from method and write SELF (unless CREATE, Debug, FreeObject, or Free)}
000882  {$ENDC}
000883
000884
000885          (* ======================  RULES FOR WRITING A Fields FUNCTION  ===================
000886
000887           This function must be defined in every class until the compiler generates this info automatically!
000888
000889              PROCEDURE TWhatever.Fields{(PROCEDURE Field(nameAndType: S255))};
000890              BEGIN  {THE FIELDS MUST BE LISTED IN DECLARED ORDER, NONE OMITTED AND NONE ADDED}
000891                {Tell the superclass first (unnecessary if it is TObject)}
000892                  SUPERSELF.Fields(Field);
000893                {The following type names are recognized by the parser}
000894                  Field('flag: BOOLEAN');
000895                  Field('coCode: Byte');
000896                  Field('inputChar: CHAR');
000897                  Field('version: INTEGER');
000898                  Field('width: LONGINT');
000899                  Field('viewLPt: LPoint');
000900                  Field('boundLRect: LRect');
000901                  Field('size: Point');
000902                  Field('ptr: Ptr');
000903                  Field('boundRect: Rect');
000904                  Field('someName: STRING[100]');
000905                {If the last field is a Byte or a BOOLEAN, force padding to a word boundary by...
000906                  Field('');
000907                {Every Registered Class name is recognized}
```

```
000908                Field('miscObj: TObject');
000909                Field('myPanel: TPanel');
000910                Field('mySel: TMySelection');
000911                Field('appSpecific: TAppSpecific');
000912              {You may report more than one field in a single call to reduce code space}
000913                Field('boundLRect: LRect; size: Point; ptr: Ptr; mySel: TMySelection');
000914              {Unpacked invariant RECORDs are recognized}
000915                Field('info: RECORD version: INTEGER; size: Point END');
000916              {If the record has variants, select among them before calling Field()}
000917                CASE SELF.variant OF
000918                    flavor1: Field('RECORD version: INTEGER; size: Point END');
000919                    flavor2: Field('RECORD viewLPt: LPoint END');
000920                    END;
000921              {Unpacked ARRAYs with literal bounds are recognized}
000922                Field('desc: ARRAY [1..99] OF RECORD version: INTEGER; id: ARRAY [1..2] OF CHAR END');
000923              {Other constructs and type names are NOT recognized; substitute one of the above forms}
000924              {As a last resort, use ARRAY [1..SIZEOF(SELF.fieldName)] OF Byte}
000925            END;
000926        *)
000927
000928  IMPLEMENTATION
000929
000930  {$I LIBTK/UOBJECT2.text}  {Objects, Classes, Streams, and Resources}
000931  {$I LIBTK/UOBJECT3.text}  {Arrays and Lists}
000932  {$I LIBTK/UOBJECT4.text}  {Debugger}
000933
000934  (**********
000935  {$I UOBJECT2.text}  {Objects, Classes, Streams, and Resources}
000936  {$I UOBJECT3.text}  {Arrays and Lists}
000937  {$I UOBJECT4.text}  {Debugger}
000938  **********)
000939
000940  END.
000941
000942

End of File -- Lines: 942 Characters: 42428
```

```
================================================================================================
FILE: "LIBTK/UOBJECT2.TEXT"
================================================================================================

000001  {INCLUDE FILE UOBJECT3 -- OBJECTS, CLASSES, RESOURCES, AND STREAMS}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004  {changed 05/01 1503 Changes to allow people to use Clascal on the Workshop}
000005
000006  {Segments: SgCLAini(tialize and Terminate), SgCLAres(ident), SgCLAc(o)ld, SgCLAdbg}
000007
000008  {$%+}
000009
000010  {$IFC fRngObject}
000011  {$R+}
000012  {$ELSEC}
000013  {$R-}
000014  {$ENDC}
000015
000016  {$IFC fSymObject}
000017  {$D+}
000018  {$ELSEC}
000019  {$D-}
000020  {$ENDC}
000021
000022  CONST
000023
000024      trLevMemory = 60;
000025          { The std value of keyPresLimit, overridable by chaging the variable keyPresLimit }
000026      stdKeyPresLimit = 10;
000027      maxTallies = 3000; { < 32K DIV SIZEOF(TTally) }
000028
000029  TYPE
000030
000031      S16 = STRING[16];
000032
000033      TPS8 = ^S8;
000034      TPByte = ^Byte;
000035      TPAOC = PACKED ARRAY[1..32767] OF CHAR;
000036      TpPAOC = ^TPAOC;
000037
000038      TppINTEGER = ^TpINTEGER;
000039
000040      TPObject = ^TObject;
000041
000042      TPPathName = ^PathName;
000043      TPEName = ^E_Name;
```

```
000044
000045      TRecycleChain = RECORD
000046                      classPtr:   TClass;
000047                      chainLink:  TObject;
000048                      END;
000049      TPRecycleChain = ^TRecycleChain;
000050      THRecycleChain = ^TPRecycleChain;
000051
000052      UnsignedByte = 0..255;
000053
000054      TTypeCode = (yBoolean, yHexByte, yByte, yChar, yHexInteger, yInteger, yLongInt, yLongReal,
000055                   yLPoint, yLRect, yObject, yPoint, yPtr, yReal, yRect, yString, yArray);
000056
000057    {We can't USE Unit UDRAW because it USES us; these are needed in EXTERNAL decls below for KitBug}
000058      FakePoint = RECORD v, h: INTEGER END;
000059      FakeRect = RECORD top, left, bottom, right: INTEGER END;
000060      FakeLPoint = RECORD v, h: LONGINT END;
000061      FakeLRect = RECORD top, left, bottom, right: LONGINT END;
000062
000063  {$IFC LibraryVersion < 20}
000064    { The following definitions come from PasLibCall and PPasLibC; if those files change, these
000065                will have to be changed too !!!! }
000066      dsProcCode = (dsResProg, dsSoftPwbtn, dsPrintDev, dsSetGPrefix, dsEnbDisk);
000067
000068      dsProcParam = record
000069          case ProcCode : dsProcCode of
000070                  dsResProg : (RProcessId : longint);
000071                  dsSoftPwbtn  : (SPButton : boolean);
000072                  dsPrintDev   : (PrDevice : e_name);
000073                  dsSetGPrefix : (errnum : INTEGER;
000074                                  prefix : pathname);
000075                  dsEnbDisk    : (DiskEvent : boolean);
000076                  end;
000077  {$ENDC}
000078
000079    {Tallying}
000080
000081      TTally = RECORD
000082          count:          INTEGER;
000083          microseconds:   LONGINT;
000084          epPC:           LONGINT;
000085          END;
000086
000087      TTallyArray = ARRAY [1..maxTallies] OF TTally;
000088
000089      TDTallyArray  = RECORD
000090          header: TArrayHeader;
000091          recs:   TTallyArray;    {name "recs" must be different from "records" in THIdxArray}
```

```
000092          END;
000093      TPDTallyArray = ^TDTallyArray;
000094      THTallies  = ^TPDTallyArray;        {An alias for a TArray of TTally}
000095
000096    {Version Conversion Types}
000097
000098      TDClasses = RECORD
000099          header:         TArrayHeader;
000100          records:        TClassArray;
000101          END;
000102      TPDClasses = ^TDClasses;
000103      THClasses = ^TPDClasses;            {An alias for a TArray of TClassInfo}
000104
000105      TDSTables = RECORD
000106          header:         TArrayHeader;
000107          records:        TSTableArray;
000108          END;
000109      TPDSTables = ^TDSTables;
000110      THSTables = ^TPDSTables;            {An alias for a TArray of TPSliceTable}
000111
000112      TDAuthorArray  = RECORD
000113          header:         TArrayHeader;
000114          records:        TAuthorArray;
000115          END;
000116      TPDAuthorArray = ^TDAuthorArray;
000117      THAuthors = ^TPDAuthorArray;    {An alias for a TArray of TA32 (company and author)}
000118
000119      TDAliasArray  = RECORD
000120          header:         TArrayHeader;
000121          records:        TAliasArray;
000122          END;
000123      TPDAliasArray = ^TDAliasArray;
000124      THAliases  = ^TPDAliasArray;     {An alias for a TArray of TA8 (class alias)}
000125
000126      TIdxArray = ARRAY [1..16000] OF INTEGER;
000127
000128      TDIdxArray  = RECORD
000129          header:         TArrayHeader;
000130          records:        TIdxArray;
000131          END;
000132      TPDIdxArray = ^TDIdxArray;
000133      THIdxArray = ^TPDIdxArray;          {An alias for a TArray of INTEGER}
000134
000135      TWorld = RECORD
000136          hExClasses:     THClasses;      {hExClasses^^ .records[i] is the TClassInfo of class no. i}
000137          hExSTables:     THSTables;      {hExSTables^^ .records[i] is the TPSliceTable of class no. i}
000138          hExAuthors:     THAuthors;      {hExAuthors^^ .records[i] is the i'th companyAndAuthor encountered}
000139          hExAliases:     THAliases;      {hExAliases^^ .records[i] is the i'th classAlias encountered}
```

```
000140          END;
000141
000142  VAR
000143
000144      hMyClasses:         THClasses;      {hMyClasses^^ .records[i] is the TClassInfo of class no. i}
000145      hMySTables:         THSTables;      {hMySTables^^ .records[i] is the TPSliceTable of class no. i}
000146      hMyAuthors:         THAuthors;      {hMyAuthors^^ .records[i] is the i'th companyAndAuthor encountered}
000147      hMyAliases:         THAliases;      {hMyAliases^^ .records[i] is the i'th classAlias encountered}
000148      hMyHashName:        THIdxArray;     {hMyHashName^^.records[hashIndex] is 0 or the index i of a class}
000149
000150      cObject:            TClass;         {The TClass of TObject}
000151
000152  {$IFC compatibleLists} {For TDynamicArray.Class and TIndexList.Class}
000153      cArray:             TClass;         {The TClass of TArray}
000154      cList:              TClass;         {The TClass of TList}
000155  {$ENDC}
000156
000157      availListScanner:   TListScanner;   {Heads of preallocated Scanner chains}
000158      availArrayScanner:  TArrayScanner;
000159      availStringScanner: TStringScanner;
000160
000161      {$IFC fTrace}
000162      fTraceEnabled, fTraceSelf, fTraceClass: BOOLEAN;
000163        { Current method nesting level }
000164      tabLevel:               INTEGER;
000165        { So EP calls don't have to pass a trace level parameter, it is saved here on the corresponding
000166          BP call. }
000167      traceLevels:            ARRAY [0..trLevMemory] OF INTEGER;          {indexed by tabLevel}
000168        { So EP can check for matching BP. }
000169      traceFrames:            ARRAY [0..trLevMemory] OF LONGINT;          {indexed by tabLevel}
000170        { To time procedure durations. }
000171
000172      traceTimes:             ARRAY [0..trLevMemory] OF LONGINT;          {indexed by tabLevel}
000173        { kpcntr counts number of times AKeyPress has been called and only calls KeyPress every
000174          keyPresLimit times for performance reasons. }
000175
000176      kpcntr:                 INTEGER;
000177        { TRUE IF returning to main screen after leaving debugger }
000178      returnToMain:           BOOLEAN;
000179        { traceCount of 0 -> no tracing, traceCount of 1 means you have traced through defTraceCount methods
000180          so time to enter the debugger. }
000181      traceCount, defTraceCount: INTEGER;
000182        { Set with the Level command }
000183      curTraceLevel:          INTEGER;
000184        { Break when you come to one of these methods }
000185      breakMethods:           ARRAY [1..maxBreaks] OF RECORD
000186                                                          brClass, brMethod: S8;
000187                                                      END;
```

```
000188        { The number of valid break methods currently active }
000189      breakMCount:            INTEGER;
000190        { TRUE IF showing the debugger prompt }
000191      showPrompt:             BOOLEAN;
000192        { TRUE if BP is tallying procedure calls }
000193      tallyingCalls:          BOOLEAN;
000194        { A hash table if tallyingCalls }
000195      tallies:                THTallies;
000196
000197  (*  tallyOverhead:          LONGINT;    {usual time spent calling and returning from BP, EP, or Tally} *)
000198  (*  debugTime:              LONGINT;    {cumulative time spend in BP and EP since tallying started} *)
000199      startTime:              LONGINT;    {when tallying started}
000200      stopTime:               LONGINT;    {when tallying last paused}
000201      segNames:               TArray{[1..127] OF S8};
000202
000203        { Used to avoid break point checking on methods we have already checked }
000204      lastBpPc:               LONGINT;
000205      lastEpPc:               LONGINT;
000206      {$ENDC}
000207
000208
000209  { ================================== EXTERNAL AND FORWARD PROCEDURES ================================== }
000210
000211
000212  {$IFC LibraryVersion < 20}
000213    {So we don't need to use PasLibCall or PPasLibC; this may have to change if those .OBJ files change !!!!}
000214      PROCEDURE OutputRedirect (VAR errnum : INTEGER; VAR outfile : pathname; stopoutput : BOOLEAN); EXTERNAL;
000215      PROCEDURE DSPaslibCall (VAR ProcParam : dsProcParam); EXTERNAL;
000216  {$ENDC}
000217
000218    {We can't USE Unit UDRAW because it USES us}
000219      PROCEDURE InitErrorAbort(error: INTEGER); EXTERNAL;
000220      PROCEDURE TrmntExceptionHandler; EXTERNAL;
000221
000222      {$IFC fDbgObject}
000223      FUNCTION  BindHeap(activeVsClip, doBind: BOOLEAN): THeap; EXTERNAL;
000224      {$ENDC}
000225
000226    {We can't USE Unit UDRAW because it USES us}
000227      PROCEDURE PointToStr(pt: FakePoint; str: TPstring); EXTERNAL;
000228      PROCEDURE RectToStr(r: FakeRect; str: TPstring); EXTERNAL;
000229      PROCEDURE LPointToStr(pt: FakeLPoint; str: TPstring); EXTERNAL;
000230      PROCEDURE LRectToStr(r: FakeLRect; str: TPstring); EXTERNAL;
000231
000232    {We can't USE Unit Storage because of type name conflicts (Ptr, Handle, ProcPtr)}
000233      PROCEDURE SetHeap(heap: THeap); EXTERNAL;
000234      PROCEDURE GetHeap(VAR heap: THeap); EXTERNAL;
000235
```

```
000236    {We can't USE Unit QuickDraw because we can't use Storage; nor WM without using QuickDraw; nor UDraw, so...}
000237      PROCEDURE InitQDWM; EXTERNAL; {in UDraw}
000238      PROCEDURE DrawText(textBuf: TpINTEGER; firstByte, byteCount: INTEGER); EXTERNAL;
000239      FUNCTION  TextWidth(textBuf: TpINTEGER; firstByte, byteCount: INTEGER): INTEGER; EXTERNAL;
000240      PROCEDURE DrawLText(textBuf: TpINTEGER; firstByte, byteCount: INTEGER); EXTERNAL;
000241
000242    {The rest are assembler routines in XFER and ARE declared in the INTERFACE of this unit}
000243      FUNCTION  LIntAndLInt(i, j: LONGINT): LONGINT; EXTERNAL;
000244      FUNCTION  LIntOrLInt(i, j: LONGINT): LONGINT; EXTERNAL;
000245      FUNCTION  LIntXorLInt(i, j: LONGINT): LONGINT; EXTERNAL;
000246      PROCEDURE XferLeft(source, dest: Ptr; nBytes: INTEGER); EXTERNAL;
000247      PROCEDURE XferRight(source, dest: Ptr; nBytes: INTEGER); EXTERNAL;
000248      FUNCTION  EqualBytes(source, dest: Ptr; nBytes: INTEGER): BOOLEAN; EXTERNAL;
000249
000250    {The rest are assembler routines in CLASLIB and are NOT declared in the INTERFACE of this unit}
000251      FUNCTION  %_GetA5: LONGINT; EXTERNAL;
000252      PROCEDURE %_GoLisabug; EXTERNAL;
000253
000254    {Forward}
000255      {$IFC fDebugMethods}
000256      PROCEDURE WriteDRecord(numLevels: INTEGER; hDRecord: Handle; posInDRecord: INTEGER;
000257                             PROCEDURE SupplyFields(PROCEDURE Field(nameAndType: S255))); FORWARD;
000258      {$ENDC}
000259
000260
000261  { ================================== COLD UTILITIES ================================== }
000262
000263
000264
000265
000266  {$S SgCLAcld}
000267  FUNCTION MakeIdxArray(numElements: INTEGER; sparse: BOOLEAN): THIdxArray;
000268      VAR anArray:    TArray;
000269          i:          INTEGER;
000270  BEGIN
000271      {$IFC fMaxTrace}BP(1);{$ENDC}
000272      {$IFC fMaxTrace}EP;{$ENDC}
000273      IF sparse THEN
000274          numElements := (((numElements + 6) * 4) DIV 3);
000275      anArray := TArray.CREATE(NIL, mainHeap, numElements, SIZEOF(INTEGER));
000276      anArray.InsNullsAt(1, numElements);
000277      MakeIdxArray := THIdxArray(anArray);
000278  (*****
000279      hArray := THIdxArray(TDynamicArray.CREATE(NIL, mainHeap, SIZEOF(INTEGER), numElements));
000280      FOR i := 1 TO numElements DO
000281          hArray^^.records[i] := 0;
000282      MakeIdxArray := hArray;
000283  *****)
```

```
000284  END;
000285
000286
000287  {$S SgCLAcld}
000288  PROCEDURE EachObject(heap: THeap; PROCEDURE DoToObject(object: TObject));
000289      VAR  hz:          THz;       { The heap as a UnitHz type }
000290           mpFirst:     LONGINT;   { The address of the first master pointer }
000291           mpLast:      LONGINT;   { The address of the last master pointer }
000292           mpIndex:     LONGINT;   { An index variable used for stepping through the master pointers }
000293           mp:          LONGINT;   { the value of the master pointer at mpIndex }
000294  BEGIN
000295      {$IFC fMaxTrace}BP(1);{$ENDC}
000296      {$IFC fMaxTrace}EP;{$ENDC}
000297      hz := THz(heap);
000298      mpFirst := ORD(@hz^.argpPool);
000299      mpLast := mpFirst + 4 * ((hz^.ipPoolMac) - 1);
000300
000301    {Step through each master pointer in heap}
000302      mpIndex := mpFirst;
000303      WHILE mpIndex <= mpLast DO
000304          BEGIN
000305          mp := ORD(Handle(mpIndex)^);
000306          IF NOT (((mp >= mpFirst) AND (mp <= mpLast)) OR (mp = 1)) THEN  {not on the free list}
000307             DoToObject(POINTER(ORD(mpIndex))); { Pass it to DoToObject as a TObject, but don't coerce
000308                                                  directly to a TObject because of run-time checking. }
000309          mpIndex := mpIndex + 4;          { advance to the next master pointer }
000310          END;
000311  END;
000312
000313
000314  { ==================================== HOT UTILITIES ====================================== }
000315
000316
000317
000318
000319  {$S sHotUtil}
000320  FUNCTION  Min(i, j: LONGINT): LONGINT;
000321  BEGIN
000322      {$IFC fMaxTrace}BP(1);{$ENDC}
000323      IF i < j THEN
000324          Min := i
000325      ELSE
000326          Min := j;
000327      {$IFC fMaxTrace}EP;{$ENDC}
000328  END;
000329
000330
000331  {$S sHotUtil}
```

```
000332  FUNCTION  Max(i, j: LONGINT): LONGINT;
000333  BEGIN
000334      {$IFC fMaxTrace}BP(1);{$ENDC}
000335      IF i > j THEN
000336          Max := i
000337      ELSE
000338          Max := j;
000339      {$IFC fMaxTrace}EP;{$ENDC}
000340  END;
000341
000342
000343  {$S sHotUtil}
000344  PROCEDURE LatestError(newError: INTEGER; VAR previousError: INTEGER);
000345  BEGIN
000346      {$IFC fMaxTrace}BP(1);{$ENDC}
000347      IF ((newError > 0) AND (previousError <= 0) OR
000348          (newError < 0) AND (previousError = 0)) THEN
000349         previousError := newError;
000350      {$IFC fMaxTrace}EP;{$ENDC}
000351  END;
000352
000353
000354  {$S sHotUtil}
000355  FUNCTION ClassPtr(hndl: Handle): TClass;
000356      VAR stp: RECORD
000357                   CASE INTEGER OF
000358                       1: (asLong: LONGINT);
000359                       2: (asBytes: PACKED ARRAY [0..3] OF TByte);
000360                       3: (asClass: TClass);
000361                       END;
000362  BEGIN
000363      {$IFC fMaxTrace}BP(1);{$ENDC}
000364      {$IFC fMaxTrace}EP;{$ENDC}
000365      stp.asLong := hndl^^;
000366      stp.asBytes[0] := 0;
000367      ClassPtr := stp.asClass;
000368  END;
000369
000370
000371  {$S sUtil}
000372  PROCEDURE LIntToHex(decNumber: LONGINT; hexNumber: TPString);
000373              {NOTE: hexNumber must be >= 8 characters, regardless of size of decNumber}
000374
000375  (* This PROCEDURE accepts a binary LONGINT, decNumber, and returns the equivalent hexadecimal *)
000376  (* number by means of the output parameter hexNumber.  Note that if the equivalent hexadecimal number is *)
000377  (* of a sufficiently small magnitude that it does not require all of the digits in the hex field to be *)
000378  (* expressed (e.g. if 8 digits are allocated in the hex field and the hex number if 58A7, which is only *)
000379  (* 4 digits), then the hexadecimal number will be right-justified with leading zeros to pad the field.  So, *)
```

```
000380  (* for example, 58A7 will be returned as 000058A7 if 8 digits are allocated for hexadecimal numbers via the *)
000381  (* constant hexFieldSize.   To change the number of digits in the hex field, change the constant *)
000382  (* hexFieldSize. *)
000383
000384  {NOTE: many users of LIntToHex pass in a pointer to a variable declared as S8; therefore, it is important
000385              that LIntToHex not return more than 8 digits }
000386
000387  CONST
000388      hexFieldSize = 8;  (* the number of digits which are to appear in a hexadecimal field;  leading zeros   *)
000389                         (* may be used to pad small hexadecimal numbers (e.g. if hexFieldSize is 8, then the
000390                         (* hex number FA9 would appear as 00000FA9) *)
000391
000392  VAR hexDigits:  S16;                (* a list which is to contain all hexadecimal digits *)
000393      i:          1..hexFieldSize;    (* a variable for indexing individual digits of the hex number's field *)
000394      fudge:      INTEGER;
000395  BEGIN
000396      {$IFC fMaxTrace}BP(1);{$ENDC}
000397      hexDigits := '0123456789ABCDEF';   (* Initialize the list of hexadecimal digits *)
000398  {$R-}hexNumber^[0] := CHR(hexFieldSize); {$IFC fRngObject}{$R+}{$ENDC}
000399
000400      IF decNumber < 0 THEN
000401          BEGIN
000402          fudge := 16;                {reverse hexDigit indexes}
000403          decNumber := decNumber + 1; {correct for two's complement}
000404          END
000405      ELSE
000406          fudge := 1;
000407
000408      FOR i := hexFieldSize DOWNTO 1 DO
000409          BEGIN
000410          hexNumber^[i] := hexDigits[(decNumber MOD 16) + fudge];
000411          decNumber := decNumber DIV 16;
000412          END;
000413      {$IFC fMaxTrace}EP;{$ENDC}
000414  END;   (* LIntToHex *)
000415
000416
000417  {$S sUtil}
000418  PROCEDURE LIntToStr(decNumber: LONGINT; str: TPString);
000419          {NOTE: str must be >= 11 characters, regardless of size of number}
000420      VAR neg : BOOLEAN;
000421          pos : INTEGER;
000422      BEGIN
000423      {$IFC fMaxTrace}BP(1);{$ENDC}
000424    {$R-} str^[0] := CHR(11); {$IFC fRngObject} {$R+} {$ENDC}
000425
000426          pos := 11;
000427          neg := (decNumber < 0);
```

```
000428          decNumber := ABS (decNumber);
000429
000430          REPEAT
000431              str^[pos] := CHR(ORD('0') + (decNumber MOD 10));
000432              pos := pos - 1;
000433              decNumber  := decNumber DIV 10;
000434          UNTIL decNumber = 0;
000435
000436          IF neg THEN
000437              BEGIN
000438              str^[pos] := '-';
000439              pos := pos - 1;
000440              END;
000441
000442          DELETE (str^, 1, pos);
000443      {$IFC fMaxTrace}EP;{$ENDC}
000444      END;
000445
000446
000447  {$S sUtil}
000448  PROCEDURE IntToStr(decNumber: INTEGER; str: TPString);
000449              {NOTE: str must be >= 6 characters (sign + 5 digits), regardless of size of decNumber}
000450      VAR s11:    STRING[11];
000451      BEGIN
000452      {$IFC fMaxTrace}BP(1);{$ENDC}
000453          LIntToStr(decNumber, @s11);
000454          XferLeft(Ptr(@s11), Ptr(str), Length(s11) + 1); { str length + 1 size byte }
000455      {$IFC fMaxTrace}EP;{$ENDC}
000456      END;
000457
000458
000459  {$S sUtil}
000460  PROCEDURE HexStrToLInt(hexString: TPString; VAR decNumber: LONGINT; VAR result: TConvResult);
000461
000462  (* This PROCEDURE accepts a STRING of hexadecimal digits, hexString, and returns a long-INTEGER decimal  *)
000463  (* equivalent by means of the variable parameter decNumber.  Information concerning the acceptability of *)
000464  (* the hexadecimal STRING is returned via the variable parameter result.                                 *)
000465  (*     Note that this PROCEDURE ignores any leading or trailing blanks which may be present in the given  *)
000466  (* hexString, and the presence of lower-case hexadecimal digits in the hex STRING does not adversely     *)
000467  (* affect conversion.  Also, if the first non-blank character of the STRING is a dollar sign, then that   *)
000468  (* dollar sign is ignored and not considered during conversion (it is, effectively, deleted from the      *)
000469  (* STRING).                                                                                             *)
000470
000471  VAR numDigits:  0..255;    (* The number of digits in the hex STRING *)
000472      digit:      CHAR;
000473      i:          INTEGER;   (* index variable *)
000474      digitValue: INTEGER;   (* index variable *)
000475      hexDigits:  S16;       (* an array which is to contain a list of hexadecimal digits *)
```

```
000476  BEGIN   (* HexStrToLInt *)
000477      {$IFC fMaxTrace}BP(1);{$ENDC}
000478
000479        (* Delete any trailing blanks *)
000480      TrimBlanks(POINTER(ORD(hexstring)));
000481
000482      { Remove any leading zeros, except keep at least 1 digit; also, remove any leading $ }
000483      IF Length(hexString^) > 0 THEN
000484          WHILE ((Length(hexString^) > 1) AND (hexString^[1] = '0')) OR (hexString^[1] = '$') DO
000485              Delete(hexString^, 1, 1);
000486
000487      numDigits := Length(hexString^);
000488
000489      decNumber := 0;
000490
000491      IF numDigits = 0 THEN   (* if the given hex STRING is empty... *)
000492          result := cvNoNumber
000493      ELSE
000494      IF Length (hexString^) > 8 THEN  (* if can't fit in LONGINT *)
000495          result := cvOverflow
000496      ELSE
000497          result := cvValid;   (* innocent until proven guilty *)
000498
000499      FOR i := 1 TO numDigits DO
000500          BEGIN
000501          digit := hexString^[i];
000502          IF digit IN ['0'..'9'] THEN
000503              digitValue := ORD(digit) - ORD('0')
000504          ELSE
000505          IF digit IN ['A'..'F'] THEN
000506              digitValue := ORD(digit) - ORD('A') + 10
000507          ELSE
000508          IF digit IN ['a'..'f'] THEN
000509              digitValue := ORD(digit) - ORD('a') + 10
000510          ELSE
000511              BEGIN
000512              digitValue := 0;
000513              result := cvBadNumber;
000514              END;
000515          decNumber := decNumber * 16 + digitValue;
000516          END;
000517      {$IFC fMaxTrace}EP;{$ENDC}
000518  END;   (* HexStrToLInt *)
000519
000520
000521  {$S sUtil}
000522  PROCEDURE StrToLInt(str: TPString; VAR decNumber: LONGINT; VAR result: TConvResult);
000523      LABEL   1;
```

```
000524
000525      VAR s:      S255;
000526          pos:    INTEGER;
000527          neg:    BOOLEAN;
000528  BEGIN
000529      {$IFC fMaxTrace}BP(1);{$ENDC}
000530      result := cvValid;
000531      XferLeft(Ptr(str), Ptr(@s), Length(str^) + 1);
000532      TrimBlanks(@s);
000533
000534      decNumber := 0;
000535      neg := FALSE;
000536
000537      IF s='' THEN
000538          result := cvNoNumber
000539      ELSE IF (s[1]='-') OR (s[1]='+') THEN
000540          BEGIN
000541          neg := s[1] = '-';
000542          Delete(s, 1, 1);
000543          IF s='' THEN
000544              result := cvBadNumber;
000545          END;
000546
000547      pos := 1;
000548      WHILE pos <= Length(s) DO
000549          BEGIN
000550          IF ('0' > s[pos]) OR (s[pos] > '9') THEN {invalid numeric character}
000551              BEGIN
000552              result := cvBadNumber;
000553              GOTO 1;
000554              END;
000555
000556          {check for overflow}
000557          IF pos > 10 THEN {more than 10 digits guarantees an overflow}
000558              BEGIN
000559              result := cvOverflow;
000560              GOTO 1;
000561              END;
000562          IF pos = 10 THEN
000563              IF ORD(s[pos]) > ORD('7') THEN
000564                  IF decNumber > 214748363 THEN
000565                      BEGIN
000566                      result := cvOverflow;
000567                      GOTO 1;
000568                      END
000569                  ELSE
000570                      { okay }
000571              ELSE { 10th digit is 7 or less }
```

```
000572                  IF decNumber > 214748364 THEN
000573                      BEGIN
000574                      result := cvOverflow;
000575                      GOTO 1;
000576                      END;
000577
000578          decNumber := (10 * decNumber) + (ORD(s[pos]) - ORD('0'));
000579          pos := pos + 1;
000580          END;
000581
000582      IF neg THEN
000583          decNumber := -decNumber;
000584  1:
000585      {$IFC fMaxTrace}EP;{$ENDC}
000586  END;
000587
000588
000589  {$S sUtil}
000590  PROCEDURE StrToInt(str: TPString; VAR decNumber: INTEGER; VAR result: TConvResult);
000591      VAR l:  LONGINT;
000592  BEGIN
000593      {$IFC fMaxTrace}BP(1);{$ENDC}
000594  {$IFC fDbgObject}
000595  {$0V+} {make sure we don't screw up}
000596  {$ENDC}
000597      StrToLint(str, l, result);
000598      IF result = cvValid THEN
000599          IF (l < -MAXINT-1) OR (l > MAXINT) THEN
000600              result := cvOverflow
000601          ELSE
000602              decNumber := INTEGER(l);
000603      {$IFC fMaxTrace}EP;{$ENDC}
000604  END;
000605
000606
000607  {$S sUtil}
000608  PROCEDURE TrimBlanks(str: TPString);
000609      LABEL
000610          1, 10;
000611
000612      CONST
000613          tabCh = CHR(9);
000614
000615      VAR i:      INTEGER;
000616
000617  BEGIN
000618      {$IFC fMaxTrace}BP(1);{$ENDC}
000619      i := 1;
```

```
000620      WHILE i <= Length(str^) DO
000621          BEGIN
000622          IF str^[i] <> ' ' THEN
000623              IF str^[i] <> tabCh THEN
000624                  BEGIN {delete all the leading stuff we have found}
000625                  Delete(str^, 1, i-1);
000626                  GOTO 1;
000627                  END;
000628          i := i + 1;
000629          END;
000630
000631      { we fell thru -- either '' or all blanks or tabs }
000632      str^ := '';
000633      GOTO 10;
000634
000635  1:  {now trim the trailing blanks}
000636
000637      i := Length(str^);
000638      WHILE i > 0 DO
000639          BEGIN
000640          IF (str^[i] = ' ') OR (str^[i] = tabCh) THEN
000641              Delete(str^, i, 1)
000642          ELSE
000643              GOTO 10;
000644          i := i - 1;
000645          END;
000646  10:
000647      {$IFC fMaxTrace}EP;{$ENDC}
000648  END;
000649
000650
000651  {$S sUtil}
000652  FUNCTION  CharUpperCased(ch: CHAR): CHAR;
000653  BEGIN
000654      {$IFC fMaxTrace}BP(1);{$ENDC}
000655      CharUpperCased := ch;
000656      IF 'a' <= ch THEN
000657          IF ch <= 'z' THEN
000658              CharUpperCased := CHR(ORD(ch) - 32);
000659      {$IFC fMaxTrace}EP;{$ENDC}
000660  END;
000661
000662
000663  {$S sUtil}
000664  PROCEDURE StrUpperCase(str: TPString);
000665      VAR i:  INTEGER;
000666  BEGIN
000667      {$IFC fMaxTrace}BP(1);{$ENDC}
```

```
000668      i := Length(str^);
000669      WHILE i > 0 DO
000670          BEGIN
000671          str^[i] := CharUpperCased(str^[i]);
000672          i := i - 1;
000673          END;
000674      {$IFC fMaxTrace}EP;{$ENDC}
000675  END;
000676
000677
000678  {$S sUtil}
000679  PROCEDURE SplitFilePath(VAR fullPath, itsCatalog, itsFilePart: TFilePath);
000680      LABEL   1;
000681      VAR i:  INTEGER;
000682  BEGIN
000683      {$IFC fMaxTrace}BP(1);{$ENDC}
000684      itsCatalog := '';
000685      itsFilePart := fullPath;
000686
000687      FOR i := Length(itsFilePart) DOWNTO 1 DO
000688          IF itsFilePart[i] = '-' THEN
000689              BEGIN
000690              itsCatalog := COPY(itsFilePart, 1, i);
000691              DELETE(itsFilePart, 1, i);
000692              GOTO 1;
000693              END;
000694  1:
000695      {$IFC fMaxTrace}EP;{$ENDC}
000696  END;
000697
000698
000699  {$S sStartup}
000700  PROCEDURE  SetCp(object: TObject; itsClass: TClass);
000701      VAR index:  INTEGER;
000702  BEGIN
000703      {$IFC fMaxTrace}BP(1);{$ENDC}
000704      Handle(object)^^ := ORD(itsClass);          {Install slice table pointer}
000705      index := CiOfCp(TPSliceTable(itsClass));    {Determine its class index}
000706      IF index < 256 THEN                         {If it will fit in a byte, store it...}
000707  {$R-}   TPByte(Handle(object)^)^ := index;      {...to speed version conversion (cf ConvertHeap: FindClasses)}
000708  {$IFC fRngObject}{$R+}{$ENDC}
000709      {$IFC fMaxTrace}EP;{$ENDC}
000710  END;
000711
000712
000713  {$S sStartup}
000714  FUNCTION  NewDynObject(heap: THeap; itsClass: TClass; dynBytes: INTEGER): TObject;
000715      VAR nBytes: INTEGER;
```

```
000716          object: TObject;
000717  BEGIN
000718      {$IFC fMaxTrace}BP(1);{$ENDC}
000719      nBytes := SizeOfCp(TPSliceTable(itsClass)) + dynBytes;
000720      object := POINTER(ORD(HAllocate(THz(heap), nBytes)));    {TObject() won't work until after SetCp}
000721      IF ORD(object) = ORD(hNIL) THEN
000722          BEGIN
000723          {$IFC fDbgObject}
000724          WriteLn(CbOfHz(THz(heap)):1, ' bytes in the heap');
000725          {$ENDC}
000726          ABCBreak('NewObject: Heap full, can''t make an object of size', nBytes);
000727          END;
000728      SetCp(object, itsClass);
000729      NewDynObject := object;
000730      {$IFC fMaxTrace}EP;{$ENDC}
000731  END;
000732
000733
000734  {$S sStartup}
000735  FUNCTION  NewObject(heap: THeap; itsClass: TClass): TObject;
000736  BEGIN
000737      {$IFC fMaxTrace}BP(1);{$ENDC}
000738      NewObject := NewDynObject(heap, itsClass, 0);
000739      {$IFC fMaxTrace}EP;{$ENDC}
000740  END;
000741
000742
000743  {$S sStartup}
000744  PROCEDURE ResizeDynObject(object: TObject; newTotalBytes: INTEGER);
000745      VAR i: INTEGER;
000746  BEGIN
000747      {$IFC fMaxTrace}BP(1);{$ENDC}
000748      IF (newTotalBytes < 0) OR (newTotalBytes > (MAXINT-20)) THEN
000749          ABCBreak('New size must lie between 0 and 32K-20, not', newTotalBytes);
000750      ChangeSizeH(THz(object.Heap), TH(object), newTotalBytes);
000751      IF CbDataOfH(THz(object.Heap), TH(object)) < newTotalBytes THEN
000752         ABCBreak('ResizeDynObject: Heap full, size can''t change to', newTotalBytes);
000753      {$IFC fMaxTrace}EP;{$ENDC}
000754  END;
000755
000756
000757  {$IFC compatibileLists}
000758  FUNCTION  SubObject(super: TObject; itsClass: TClass): TObject;
000759  BEGIN
000760      ResizeDynObject(super, SizeOfCp(TPSliceTable(itsClass)));
000761      SetCP(super, itsClass);
000762      SubObject := super;
000763  END;
```

```
000764  {$ENDC}
000765
000766
000767  {$S sStartup}
000768  FUNCTION  NewOrRecycledObject(heap: THeap; itsClass: TClass; VAR chainHead: TObject): TObject;
000769  BEGIN
000770      {$IFC fMaxTrace}BP(1);{$ENDC}
000771      IF chainHead = NIL THEN
000772          NewOrRecycledObject := NewObject(heap, itsClass)
000773      ELSE
000774          BEGIN
000775          {$IFC fDbgObject}
000776          IF (chainHead.Class <> itsClass) OR (chainHead.Heap <> heap) THEN
000777              ABCBreak('NewOrRecycledObject: chainHead contains an alien object', ORD(chainHead));
000778          {$ENDC}
000779          NewOrRecycledObject := chainHead;
000780          chainHead := THRecycleChain(chainHead)^^.chainLink;
000781          END;
000782      {$IFC fMaxTrace}EP;{$ENDC}
000783  END;
000784
000785
000786  {$S sStartup}
000787  PROCEDURE RecycleObject(object: TObject; VAR chainHead: TObject);
000788      {$IFC fDbgObject}
000789      VAR chainMember: TObject;
000790      {$ENDC}
000791  BEGIN
000792      {$IFC fMaxTrace}BP(1);{$ENDC}
000793      {$IFC fDbgObject}
000794      IF object.HeapBytes < 8 THEN
000795          ABCBreak('RecycleObject: object is too small for a chainHead link', ORD(object));
000796      chainMember := chainHead;
000797      WHILE chainMember <> NIL DO
000798          BEGIN
000799          IF chainMember = object THEN
000800              ABCBreak('RecycleObject: object freed twice', ORD(object));
000801          chainMember := THRecycleChain(chainMember)^^.chainLink;
000802          END;
000803      {$ENDC}
000804      THRecycleChain(object)^^.chainLink := chainHead;
000805      chainHead := object;
000806      {$IFC fMaxTrace}EP;{$ENDC}
000807  END;
000808
000809
000810  {$S sRes}
000811  PROCEDURE Recreate(object: TObject; oldSize, newSize: INTEGER; newSTP: TPSliceTable);
```

```
000812      VAR extraPtr:   TPByte;
000813          hz:         THz;
000814          cb:         INTEGER;
000815          bk:         LONGINT;
000816  BEGIN
000817      {$IFC fMaxTrace}BP(1);{$ENDC}
000818      SetCP(object, TClass(newSTP));      {Install the new slice-table pointer}
000819      IF newSize <> oldSize THEN          {Default extra fields to 0/NIL}
000820          BEGIN
000821          hz := HzFromH(TH(object));
000822          cb := CbDataOfH(hz, TH(object));
000823          bk := ORD(Handle(object)^);
000824
000825          IF (cb > oldSize) AND (newSize < oldSize) THEN    {There is a variable-length part & we're shrinking}
000826              XferLeft(Ptr(bk + oldSize), Ptr(bk + newSize), cb - oldSize);
000827
000828          ChangeSizeH(hz, TH(object), cb + newSize - oldSize);
000829
000830          IF (cb > oldSize) AND (newSize > oldSize) THEN    {There is a variable-length part & we're expanding}
000831              XferRight(Ptr(bk + oldSize), Ptr(bk + newSize), cb - oldSize);
000832
000833          IF newSize > oldSize THEN       {Default extra fields to 0/NIL}
000834              BEGIN
000835              extraPtr := TPByte(bk + oldSize + 1);
000836              extraPtr^ := 0;         {Store one zero and let XferLeft copy it repeatedly}
000837              XferLeft(Ptr(extraPtr), Ptr(ORD(extraPtr) + 1), newSize - oldSize - 1);
000838              END;
000839          END;
000840      {$IFC fMaxTrace}EP;{$ENDC}
000841  END;
000842
000843
000844  {$S sRes}
000845  FUNCTION  Superclass(class: TClass): TClass;
000846  BEGIN
000847      {$IFC fMaxTrace}BP(1);{$ENDC}
000848  {$R-} Superclass := TClass(TPSliceTable(class)^[-1]); {$IFC fRngObject}{$R+}{$ENDC}
000849      {$IFC fMaxTrace}EP;{$ENDC}
000850  END;
000851
000852
000853  {$S sRes}
000854  FUNCTION  ClassDescendsFrom(descendant, ancestor: TClass): BOOLEAN;
000855  BEGIN
000856      {$IFC fMaxTrace}BP(1);{$ENDC}
000857      WHILE (descendant <> ancestor) AND (descendant <> NIL) DO
000858          {$R-} descendant := TClass(TPSliceTable(descendant)^[-1]); {$IFC fRngObject}{$R+}{$ENDC}
000859      ClassDescendsFrom := descendant <> NIL;
```

```
000860      {$IFC fMaxTrace}EP;{$ENDC}
000861  END;
000862
000863
000864  {$S sRes}
000865  PROCEDURE NameOfClass(class: TClass; VAR className: TClassName);
000866  BEGIN
000867      {$IFC fMaxTrace}BP(1);{$ENDC}
000868      CpToCn(TPSliceTable(class), TS8(className));
000869      {$IFC fMaxTrace}EP;{$ENDC}
000870  END;
000871
000872
000873  {$S sRes}
000874  FUNCTION  SizeOfClass(class: TClass): INTEGER;
000875  BEGIN
000876      {$IFC fMaxTrace}BP(1);{$ENDC}
000877      SizeOfClass := SizeOfCp(TPSliceTable(class));
000878      {$IFC fMaxTrace}EP;{$ENDC}
000879  END;
000880
000881
000882  {$S SgCLAres}
000883      {toInsert, return: -1 if class already there or if table full, index if a hole found}
000884      {not toInsert, return: index (> 0) if class found, -1 if not there}
000885  FUNCTION LookupName(classAlpha: TA8; toInsert: BOOLEAN): INTEGER;
000886
000887      FUNCTION CompareName(hashIndex: INTEGER): THashCompare;
000888          VAR myIndex:    INTEGER;
000889              trialName:  TS8;
000890      BEGIN
000891          myIndex := hMyHashName^^.records[hashIndex];
000892          IF myIndex = 0 THEN
000893              CompareName := cHole
000894          ELSE
000895          IF classAlpha = hMyClasses^^.records[myIndex].classAlpha THEN
000896              CompareName := cMatch
000897          ELSE
000898              CompareName := cMismatch;
000899      END;
000900
000901  BEGIN
000902      {$IFC fMaxTrace}BP(1);{$ENDC}
000903      {$IFC fMaxTrace}EP;{$ENDC}
000904      LookupName := LookupInHashArray(hMyHashName^^.header.size,
000905                                      ORD(classAlpha[2])*ORD(classAlpha[4])+ORD(classAlpha[6]),
000906                                      toInsert, CompareName);
000907  END;
```

```
000908
000909
000910  {$S SgCLAres}
000911  FUNCTION  ValidDataAddress(addr: LONGINT): BOOLEAN;
000912            {Returns TRUE iff: addr is in a data segment (stack seg doesn't qualify)
000913                      AND is it an even address
000914                      AND is it within the bounds of the data segment}
000915
000916      CONST   dsMaxSize   = $00020000; {128K}
000917
000918      VAR error:  INTEGER;
000919          refnum: INTEGER;
000920          dsInfo: dsInfoRec;
000921  BEGIN
000922      {$IFC fMaxTrace}BP(1);{$ENDC}
000923      ValidDataAddress := FALSE;
000924
000925      IF NOT ODD(addr) THEN
000926          BEGIN
000927          Info_Address(error, addr, refnum);
000928          IF error <= 0 THEN
000929              BEGIN
000930              Info_Dataseg(error, refnum, dsInfo);
000931              IF error <= 0 THEN
000932                  IF (addr MOD dsMaxSize) < dsInfo.mem_size THEN
000933                      ValidDataAddress := TRUE;
000934              END;
000935          END;
000936      {$IFC fMaxTrace}EP;{$ENDC}
000937  END;
000938
000939
000940  {$S sStartup}
000941  PROCEDURE Free(object: TObject);
000942  BEGIN
000943      {$IFC fMaxTrace}BP(1);{$ENDC}
000944      IF object <> NIL THEN
000945          object.Free;
000946      {$IFC fMaxTrace}EP;{$ENDC}
000947  END;
000948
000949
000950  {*** THE FOLLOWING TWO ROUTINES ASSUME THAT THE hHashName AND hMyClasses TABLES ARE ALWAYS AROUND ***}
000951          {*** IF THEY START SWAPPING OUT, WRITE LINEAR SEARCH ROUTINES TO REPLACE THESE ***}
000952
000953
000954  {$S SgCLAres}
000955  FUNCTION CiOfAlpha(classAlpha: TA8): INTEGER;       {convert class title TA8 to class index}
```

```
000956      VAR hashIndex:  INTEGER;
000957          i:          INTEGER;
000958  BEGIN
000959      {$IFC fMaxTrace}BP(1);{$ENDC}
000960      hashIndex := LookupName(classAlpha, FALSE);
000961      IF hashIndex <= 0 THEN
000962          CiOfAlpha := 0
000963      ELSE
000964          CiOfAlpha := hMyHashName^^.records[hashIndex];
000965      {$IFC fMaxTrace}EP;{$ENDC}
000966  END;
000967
000968
000969  {$S SgCLAres}
000970  FUNCTION CiOfCn(className: S8): INTEGER;            {convert upper-case class title S8 to class index}
000971      VAR a8: TA8;
000972  BEGIN
000973      {$IFC fMaxTrace}BP(1);{$ENDC}
000974      FillChar(a8, 8, ' ');
000975      XferLeft(Ptr(ORD(@className)+1), @a8, Length(className));
000976      CiOfCn := CiOfAlpha(a8);
000977      {$IFC fMaxTrace}EP;{$ENDC}
000978  END;
000979
000980
000981
000982  { ================================== INITIALIZATION ====================================== }
000983
000984
000985  {$S sError}
000986
000987  PROCEDURE CheckInitError(error: INTEGER);
000988  BEGIN
000989      {$IFC fMaxTrace}BP(1);{$ENDC}
000990      IF error > 0 THEN {Can only call with error > 0 before TProcess class-init has run}
000991          InitErrorAbort(error);
000992      {$IFC fMaxTrace}EP;{$ENDC}
000993  END;
000994
000995
000996  {$S sInit1}
000997  FUNCTION  NewHeap(VAR error: INTEGER; heapStart, numBytes: LONGINT; numObjects: INTEGER): THeap;
000998      VAR heap:       THeap;
000999  BEGIN
001000      {$IFC fMaxTrace}BP(1);{$ENDC}
001001      heap := THeap(HzInit(POINTER(heapStart), POINTER(heapStart+numBytes),
001002                                              NIL, numObjects, 0, POINTER(procNil),
001003                                              POINTER(procNil), POINTER(procNil), POINTER(procNil)));
```

```
001004      IF heap = POINTER(1) THEN
001005          BEGIN
001006          error := erInternal;
001007          ABCBreak('NewHeap could not make a heap of size', numBytes);
001008          heap := NIL;
001009          END
001010      ELSE
001011          error := 0;
001012
001013      NewHeap := heap;
001014      {$IFC fMaxTrace}EP;{$ENDC}
001015  END;
001016
001017
001018  {$S sInit1}
001019  FUNCTION  MakeDataSegment(VAR error, dsRefnum: INTEGER; firstTryVolume, thenTryVolume: TFilePath;
001020                                ldsn, memBytes, diskBytes: INTEGER): LONGINT;
001021      VAR startAddress:  LONGINT;
001022
001023      PROCEDURE TryMakeDataSegment(volumePart: TFilePath);
001024          VAR dsPathname: PathName;
001025      BEGIN
001026          dsPathname := Concat(volumePart, 'ds_private');
001027          Make_Dataseg(error, dsPathname, memBytes, diskBytes, dsRefnum, startAddress, ldsn, ds_private);
001028      END;
001029
001030  BEGIN
001031      {$IFC fMaxTrace}BP(1);{$ENDC}
001032      TryMakeDataSegment(firstTryVolume);
001033      IF error = 309 THEN
001034          IF firstTryVolume <> thenTryVolume THEN
001035              TryMakeDataSegment(thenTryVolume);
001036
001037      IF error >0 THEN
001038          BEGIN
001039          ABCBreak('MakeDataSegment', error);
001040          startAddress := 0;
001041          END;
001042      MakeDataSegment := startAddress;
001043      {$IFC fMaxTrace}EP;{$ENDC}
001044  END;
001045
001046
001047  {$S sInit1}
001048  PROCEDURE InitObject;
001049      VAR dsp:            DsProcParam;
001050          excepName:      T_Ex_Name;
001051          error:          INTEGER;
```

```
001052          prcsInfo:       ProcInfoRec;
001053          heapBase:       LONGINT;
001054          progVolume:     PathName;
001055  BEGIN
001056    {Until we call InitQDWM, NOTHING CAN FAIL!!!!}
001057
001058      isInitialized := FALSE;     {An interface variable set true at a higher level: e.g., by TProcess.Run}
001059      amDying := FALSE;           {An interface variable set true at a higher level when ImDying is called}
001060      wmIsInitialized := FALSE;   {An interface variable set true at a higher level: e.g., by InitQDWM}
001061
001062      {$IFC fTrace}
001063      fTraceEnabled := FALSE;
001064      fDebugRecursion := FALSE;
001065      tabLevel := -1;
001066      curTraceLevel := 1;
001067      traceCount := 0;
001068      defTraceCount := 0;
001069      breakMCount := 0;
001070      kpcntr := 0;
001071      keyPresLimit := stdKeyPresLimit;
001072      returnToMain := TRUE;
001073      showPrompt := TRUE;
001074      outputIndent := 0;
001075      currXPos := 0;
001076      tallyingCalls := FALSE;
001077      tallies := NIL;
001078      segNames := NIL;
001079      {$ENDC}
001080
001081    {Determine environment and program volume name}
001082      Info_Process(error, My_id, prcsInfo);
001083
001084      {get my volume name as '-volname-'; assumes that the OS gives us back a program name of the form:
001085                                              '-volname-progname'}
001086      Delete(prcsInfo.progPathName, 1, 1); {the first '-'}
001087      progVolume := Concat('-', Copy(prcsInfo.progPathName, 1, Pos('-', prcsInfo.progPathName)));
001088
001089
001090  {$IFC LibraryVersion <= 20}
001091    {Yu Ying has a better way to know if we are on the desktop or in the workshop, but meanwhile...}
001092      IF prcsInfo.father_Id > 1 THEN
001093          BEGIN
001094          Info_Process(error, prcsInfo.father_Id, prcsInfo);
001095
001096          {this assumes that the OS returns a program name of the form '-volname-progname'}
001097          Delete(prcsInfo.progPathName, 1, 1); {the first '-'}
001098          Delete(prcsInfo.progPathName, 1, Pos('-', prcsInfo.progPathName)); {the 'volname-'}
001099          StrUpperCased(@prcsInfo.progPathName);
```

```
001100
001101          onDesktop := prcsInfo.progPathName = 'SHELL.OFFICE SYSTEM';
001102          END
001103      ELSE
001104          BEGIN
001105          onDesktop := FALSE;
001106          END;
001107  {$ELSEC}
001108      dsp.procCode := dsGetDiskEnbF;
001109      DSPaslibCall(dsp);
001110      onDesktop := NOT dsp.diskEnbF;
001111  {$ENDC}
001112
001113      InitQDWM;   {must be the first thing before any operations that could fail;
001114                       when running on the Workshop, it also sets up the FontMgr & writeln to alternate screen.}
001115
001116      {$IFC fDbgObject}
001117      Write('Running on the ');
001118      IF onDesktop THEN
001119          WriteLn('desktop')
001120      ELSE
001121          WriteLn('workshop');
001122      {$ENDC}
001123
001124    {Declare an OS Exception Handler}
001125      excepName := 'SYS_TERMINATE';
001126      Declare_Excep_Hdl(error, excepName, @TrmntExceptionHandler);
001127      CheckInitError(error);
001128
001129      {$IFC fDbgObject}
001130      GoToXY(0,31);
001131      {$ENDC}
001132
001133    {Create data segment and heap}
001134      mainLdsn := prcsLdsn;
001135      heapBase := MakeDataSegment(error, mainDsRefnum, '', progVolume, mainLdsn, prcsDsBytes, prcsDsBytes);
001136      CheckInitError(error);
001137
001138      mainHeap := NewHeap(error, heapBase, prcsDsBytes, prcsDsBytes DIV 20);
001139      CheckInitError(error);
001140
001141      SetHeap(mainHeap);
001142  END;
001143
001144
001145  {$S sInit1}
001146  PROCEDURE UnitAuthor(companyAndAuthor: TAuthorName);                        {required once per unit}
001147      VAR a32:   TA32;
```

```
001148  BEGIN
001149      StrUpperCased(@companyAndAuthor);
001150      FillChar(a32, 32, ' ');
001151      XferLeft(Ptr(ORD(@companyAndAuthor)+1), @a32, LENGTH(companyAndAuthor));
001152      QUnitAuthor(a32);
001153  END;
001154
001155
001156  {$S sInit1}
001157  PROCEDURE ClassAuthor(companyAndAuthor: TAuthorName; classAlias: TClassName);       {optional}
001158      VAR a32:   TA32;
001159          a8:    TA8;
001160  BEGIN
001161      IF LENGTH(companyAndAuthor) > 0 THEN
001162          BEGIN
001163          StrUpperCased(@companyAndAuthor);
001164          FillChar(a32, 32, ' ');
001165          XferLeft(Ptr(ORD(@companyAndAuthor)+1), @a32, LENGTH(companyAndAuthor));
001166          QClassAuthor(a32);
001167          END;
001168
001169      IF LENGTH(classAlias) > 0 THEN
001170          BEGIN
001171          StrUpperCased(@classAlias);
001172          FillChar(a8, 8, ' ');
001173          XferLeft(Ptr(ORD(@classAlias)+1), @a8, LENGTH(classAlias));
001174          QClassAlias(a8);
001175          END;
001176  END;
001177
001178
001179  {$S sInit1}
001180  PROCEDURE ClassVersion(itsVersion, oldestItCanRead: Byte);            {optional}
001181  BEGIN
001182      IF (itsVersion < 0) OR (itsVersion > 127) OR (oldestItCanRead < 0) OR (oldestItCanRead > 127) OR
001183          (oldestItCanRead > itsVersion) THEN
001184              ABCBreak('Version numbers must be in the range 0..127 and oldestItCanRead <= itsVersion',
001185                       itsVersion);
001186      QClassVersion(itsVersion, oldestItCanRead);
001187  END;
001188
001189
001190  { ================================== VERSION CONVERSION ================================== }
001191
001192
001193  {$S SgCLAcld}
001194  PROCEDURE ConvClass(object: TObject; exWorld: TWorld; exIndex, myIndex: INTEGER);
001195  BEGIN
```

```
001196      {$IFC fMaxTrace}BP(1);{$ENDC}
001197      Recreate(object, exWorld.hExClasses^^.records[exIndex].objectSize, (*^*)
001198          hMyClasses^^.records[myIndex].objectSize, hMySTables^^.records[myIndex]); (*^*)
001199      {$IFC fMaxTrace}EP;{$ENDC}
001200  END;
001201
001202
001203  {$S SgCLAcld}
001204  FUNCTION IndexOfExClass(exWorld: TWorld; exIndex: INTEGER): INTEGER;
001205      LABEL 1,2;
001206      VAR exAuthor:   TA32;
001207          exAlias:    TA8;
001208          exAlpha:    TA8;
001209          coCode:     INTEGER;
001210          alCode:     INTEGER;
001211          index:      INTEGER;
001212  BEGIN
001213      {$IFC fMaxTrace}BP(1);{$ENDC}
001214      {$IFC fMaxTrace}EP;{$ENDC}
001215      IndexOfExClass := 0;
001216
001217      WITH exWorld, hExClasses^^.records[exIndex] DO (*^*)(* WHOLE BLOCK CHANGED *)
001218          BEGIN
001219          exAlpha := classAlpha;
001220
001221          IF classAlias = 0 THEN
001222              exAlias := classAlpha
001223          ELSE
001224              exAlias := hExAliases^^.records[classAlias];
001225
001226          IF companyAndAuthor <> 0 THEN
001227              BEGIN
001228              exAuthor := hExAuthors^^.records[companyAndAuthor];
001229              WITH hMyAuthors^^ DO
001230                  FOR coCode := 1 TO numAuthors DO
001231                      IF records[coCode] = exAuthor THEN (*^*)
001232                          GOTO 1;
001233              END;
001234          coCode := 0;
001235    1:
001236          END;
001237
001238      {If that class name is in my alias list, do it the hard way}
001239      WITH hMyAliases^^ DO
001240          FOR alCode := 1 TO numAliases DO
001241              IF records[alCode] = exAlias THEN
001242                  GOTO 2;
001243
```

```
001244      {If that class name is one of mine, too, do it the easy way}
001245      index := CiOfAlpha(exAlpha);
001246      IF index <> 0 THEN (*^*)
001247          IF hMyClasses^^.records[index].companyAndAuthor = coCode THEN
001248              BEGIN (*^*)
001249              IndexOfExClass := index;
001250              EXIT(IndexOfExClass); (*^*)
001251              END; (*^*)
001252
001253      {Different company name or never heard of that class name at all, return 0}
001254      EXIT(IndexOfExClass); (*^*)
001255
001256    2:
001257      {The hard way: exhaustive search, because we may be using different names for the same class}
001258      WITH hMyClasses^^ DO
001259          FOR index := 1 TO numClasses DO
001260              WITH records[index] DO
001261                  IF coCode = companyAndAuthor THEN
001262                      IF alCode = classAlias THEN
001263                          BEGIN
001264                          IndexOfExClass := index;
001265                          EXIT(IndexOfExClass);
001266                          END;
001267  END;
001268
001269
001270  {$S SgCLAcld}
001271  FUNCTION  NeedConversion(exClassWorld: TClassWorld; VAR olderVersion, newerVersion: BOOLEAN): BOOLEAN;
001272      VAR someDifference:     BOOLEAN;
001273          exWorld:            TWorld;
001274          numExClasses:       INTEGER;
001275          exIndex:            INTEGER;
001276          exInfo:             TClassInfo;
001277          exSize:             INTEGER;
001278          exSTP:              TPSliceTable;
001279          myIndex:            INTEGER;
001280          myInfo:             TClassInfo;
001281          mySize:             INTEGER;
001282          mySTP:              TPSliceTable;
001283  BEGIN
001284      {$IFC fMaxTrace}BP(1);{$ENDC}
001285      {$IFC fMaxTrace}EP;{$ENDC}
001286      someDifference := FALSE;
001287      olderVersion := FALSE;
001288      newerVersion := FALSE;
001289
001290      exWorld := TWorld(exClassWorld);        {Separate statement because of a compiler bug}
001291      WITH exWorld DO
```

```
001292          BEGIN
001293          numExClasses := hExClasses^^.header.size;
001294
001295          IF numClasses <> numExClasses THEN
001296              someDifference := TRUE;
001297
001298          FOR exIndex := 1 TO numExClasses DO
001299              BEGIN
001300              myIndex := IndexOfExClass(exWorld, exIndex);
001301              IF myIndex = 0 THEN
001302                  newerVersion := TRUE
001303              ELSE
001304                  BEGIN
001305                  exInfo := hExClasses^^.records[exIndex];
001306                  exSize := exInfo.objectSize;
001307                  exSTP := hExSTables^^.records[exIndex];
001308
001309                  myInfo := hMyClasses^^.records[myIndex];
001310                  mySize := myInfo.objectSize;
001311                  mySTP := hMySTables^^.records[myIndex];
001312
001313                  IF (myInfo.version < exInfo.version) OR (mySize < exSize) THEN
001314                      newerVersion := TRUE;
001315
001316                  IF (myInfo.version > exInfo.version) OR (mySize > exSize) THEN
001317                      olderVersion := TRUE;
001318
001319                  IF (mySTP <> exSTP) OR (myInfo.oldestReadableVersion <> exInfo.oldestReadableVersion) THEN
001320                      someDifference := TRUE;
001321
001322                  IF exInfo.superIndex = 0 THEN
001323                      BEGIN
001324                      IF myInfo.superIndex <> 0 THEN
001325                          newerVersion := TRUE;
001326                      END
001327                  ELSE
001328                  IF myInfo.superIndex <> IndexOfExClass(exWorld, exInfo.superIndex) THEN
001329                      newerVersion := TRUE;
001330                  END;
001331              END;
001332          END;
001333
001334      NeedConversion := someDifference OR olderVersion OR newerVersion;
001335  END;
001336
001337
001338  {$S SgCLAcld}
001339  PROCEDURE ConvertHeap(heap: THeap; exClassWorld: TClassWorld);
```

```
001340
001341      {*** VERSION CONVERSION ***
001342       Convert all the contents of heap from its classes to ours.
001343       The job is done in two passes through heap:
001344          (1) ConvertClass changes the method-table pointer of each object, and may change its size.
001345              If the object grows, extra fields are defaulted to 0/NIL.
001346          (2) ConvertFields tells each object to "Convert(oldVersion)", thus giving the application a
001347                  chance to calculate extra fields or otherwise modify the converted object.}
001348
001349      VAR exWorld:        TWorld;
001350          needPassTwo:    BOOLEAN;
001351          numExClasses:   INTEGER;
001352          hExHashSTP:     THIdxArray;
001353          hExEquivalent:  THIdxArray;
001354          exIndex:        INTEGER;
001355
001356          {toInsert, return: -1 if sliceTable already there or if table full, index if a hole found}
001357          {not toInsert, return: index (> 0) if sliceTable found, -1 if not there}
001358      FUNCTION LookupSTP(stp: TPSliceTable; toInsert: BOOLEAN): INTEGER;
001359
001360          FUNCTION CompareSTP(hashIndex: INTEGER): THashCompare;
001361              VAR myIndex:   INTEGER;
001362          BEGIN
001363              myIndex := hExHashSTP^^.records[hashIndex];
001364              IF myIndex = 0 THEN
001365                  CompareSTP := cHole
001366              ELSE
001367              IF exWorld.hExSTables^^.records[myIndex] = stp THEN
001368                  CompareSTP := cMatch
001369              ELSE
001370                  CompareSTP := cMismatch;
001371          END;
001372
001373      BEGIN
001374          LookupSTP := LookupInHashArray(hExHashSTP^^.header.size, ORD(stp), toInsert, CompareSTP);
001375      END;
001376
001377      FUNCTION EquivIndex(exIndex: INTEGER): INTEGER;
001378          VAR tblIndex:       INTEGER;
001379              myIndex:        INTEGER;
001380      BEGIN
001381          tblIndex := exIndex;
001382
001383          WITH exWorld DO
001384              WHILE tblIndex <> 0 DO
001385                  WITH hExClasses^^.records[tblIndex] DO
001386                      BEGIN
001387                      myIndex := IndexOfExClass(exWorld, tblIndex);
```

```
001388                      IF myIndex <> 0 THEN
001389                          IF version >= hMyClasses^^.records[myIndex].oldestReadableVersion THEN
001390                              BEGIN
001391                              EquivIndex := myIndex;
001392                              EXIT(EquivIndex);
001393                              END;
001394                      tblIndex := superIndex;
001395                      END;
001396
001397          {$IFC fDbgObject}
001398          ABCBreak('No common superclass', exIndex);
001399          {$ENDC}
001400          EquivIndex := 0;
001401      END;
001402
001403      FUNCTION FindClasses(object: TObject; VAR exIndex, myIndex: INTEGER; VAR moreConversion: BOOLEAN)
001404                          : BOOLEAN;
001405          {Given an object, return the original and my class index}
001406          VAR stp:            TPSliceTable;
001407              pStp:           ^TPSliceTable;(*^*)
001408              exHashIndex:    INTEGER;
001409              exVersion:      INTEGER;
001410              exSize:         INTEGER;
001411      BEGIN
001412          FindClasses := FALSE;
001413        {Determine the original class of the object from its method table ptr}
001414          stp := TPSliceTable(Handle(object)^^);
001415
001416  (*^*)
001417        {Obtain access to its high byte}
001418          pStp := @stp;
001419
001420        {stp probably has its exIndex stored in its high byte (unless > 255 or not a Clascal object)}
001421          exIndex := TPByte(pStp)^;
001422          TPByte(pStp)^ := 0;                 {So the stp comparisons below will be uncluttered}
001423
001424          IF exIndex < 0 THEN
001425              exIndex := 256 + exIndex;       {Undo sign extension caused by TPByte^}
001426
001427          IF exIndex <> 0 THEN                {It might be a class pointer}
001428              IF exIndex > numExClasses THEN
001429                  exIndex := 0                {Not a real class pointer}
001430              ELSE                            {Could not use "WITH exWorld" here because code generator balked}
001431              IF exWorld.hExSTables^^.records[exIndex] <> stp THEN
001432                  exIndex := 0;               {Not a real class pointer}
001433  (*^*){Also added next 3 comments below}
001434
001435          IF exIndex = 0 THEN                 {It is not a class pointer, or exIndex>255}
```

```
001436              IF numExClasses > 255 THEN      {It might be a class pointer after all}
001437                  BEGIN                       {Look in the hash table}
001438                  exHashIndex := LookupSTP(stp, FALSE);
001439                  IF exHashIndex <= 0 THEN    {Not a Clascal object}
001440                      Exit(FindClasses);
001441                  exIndex := hExHashSTP^^.records[exHashIndex];
001442                  END
001443              ELSE
001444                  Exit(FindClasses); {not a Clascal object}
001445
001446        {Determine the equivalent class in my process}
001447          myIndex := hExEquivalent^^.records[exIndex];
001448          FindClasses := TRUE;
001449          WITH exWorld.hExClasses^^.records[exIndex] DO
001450              BEGIN
001451              exVersion := version;
001452              exSize := objectSize;
001453              END;
001454
001455          WITH hMyClasses^^.records[myIndex] DO
001456              moreConversion := (exVersion < version) OR (exSize < objectSize);
001457
001458  (**** Replaced the following line by the preceding because it is too complicated for the
001459              Spring Release code generator:
001460
001461          WITH exWorld.hExClasses^^ DO
001462              moreConversion := (records[exIndex].version < hMyClasses^^.records[myIndex].version) OR
001463                                (records[exIndex].objectSize < hMyClasses^^.records[myIndex].objectSize);
001464  ****)
001465       END;
001466
001467      PROCEDURE ConvertClass(object: TObject);
001468          {Pass 1: Map the method-table ptr from the original to mine and change the object size}
001469          VAR exIndex:        INTEGER;
001470              myIndex:        INTEGER;
001471              moreConverson:  BOOLEAN;
001472      BEGIN
001473        {Determine both the original and my class}
001474          IF FindClasses(object, exIndex, myIndex, moreConversion) THEN
001475              BEGIN
001476            {Convert the method table pointer, change the size, default extra fields to 0/NIL}
001477              ConvClass(object, exWorld, exIndex, myIndex);
001478              IF moreConversion THEN {a second pass will be needed to let the app do special defaulting}
001479                  needPassTwo := TRUE;
001480              END;
001481      END;
001482
001483      PROCEDURE ConvertFields(object: TObject);
```

```
001484          {Pass 2: Default extra fields; a separate pass so the application can follow pointers if need be}
001485          VAR exIndex:        INTEGER;
001486              myIndex:        INTEGER;
001487              moreConverson:  BOOLEAN;
001488      BEGIN
001489        {Determine both the original and my class}
001490          IF FindClasses(object, exIndex, myIndex, moreConversion) THEN
001491              IF moreConversion THEN   {Let the app supply extra fields etc.}
001492                  object.Convert(exWorld.hExClasses^^.records[exIndex].version);
001493      END;
001494
001495  BEGIN
001496      {$IFC fMaxTrace}BP(1);{$ENDC}
001497      {$IFC fMaxTrace}EP;{$ENDC}
001498      exWorld := TWorld(exClassWorld);
001499      WITH exWorld DO
001500          BEGIN
001501          numExClasses := hExClasses^^.header.size;
001502
001503        {Make temporary arrays that will speed up reconciliation of the two worlds}
001504          hExEquivalent := MakeIdxArray(numExClasses, FALSE);
001505          FOR exIndex := 1 TO numExClasses DO
001506              hExEquivalent^^.records[exIndex] := EquivIndex(exIndex);
001507          IF numExClasses > 255 THEN
001508              BEGIN
001509              hExHashSTP := MakeIdxArray(numExClasses - 255, TRUE);
001510              FOR exIndex := 256 TO numExClasses DO
001511                  hExHashSTP^^.records[LookupSTP(hExSTables^^.records[exIndex], TRUE)] := exIndex;
001512              END;
001513          END;
001514
001515      needPassTwo := FALSE;
001516    {Pass One -- convert method table pointers (STPs)}
001517      EachObject(heap, ConvertClass);
001518    {Pass Two -- let application default extra fields}
001519      IF needPassTwo THEN
001520          EachObject(heap, ConvertFields);
001521
001522    {Free the temporary arrays}
001523      FreeH(THz(mainHeap), TH(hExEquivalent));
001524      IF numExClasses > 255 THEN (*^*)
001525          FreeH(THz(mainHeap), TH(hExHashSTP));
001526  END;
001527
001528
001529  {$S sError}
001530  PROCEDURE ClascalReason(error: INTEGER; VAR s: S255);
001531  BEGIN
```

```
001532      CASE error OF
001533          OTHERWISE s := 'Some kind of problem';      {** Need more cases **}
001534          END;
001535  END;
001536
001537
001538  {$S sInit1}
001539  PROCEDURE ClascalError(error: INTEGER); {called with error = 0 after successful Clascal initialization}
001540      VAR s:  S255;
001541          i:  INTEGER;
001542  BEGIN
001543      IF error > 0 THEN
001544          BEGIN
001545          {$IFC fDbgObject}
001546          ClascalReason(error, s);
001547          {$ENDC}
001548          IF isInitialized THEN
001549              BEGIN
001550              {$IFC fDbgObject}
001551              ABCBreak(s, error);
001552              {$ENDC}
001553              TrmntExceptionHandler;
001554              END
001555          ELSE
001556              BEGIN
001557              {$IFC fDbgObject}
001558              WriteLn('Clascal error: ', s);
001559              {$ENDC}
001560              IF wmIsInitialized THEN
001561                  InitErrorAbort(error)
001562              ELSE
001563              {$IFC fDbgObject}
001564                  %_GoLisaBug;
001565              {ELSEC}
001566                  HALT;
001567              {$ENDC}
001568              END;
001569          END
001570      ELSE
001571      IF NOT classesInitialized THEN
001572          BEGIN
001573        {*** STILL TO DO: The first time the program runs, write to the tool resource file ***}
001574
001575          {Save conversion information not obtainable from UClascal in permanent arrays}
001576
001577  (*****
001578          hMyClasses := THClasses(TDynamicArray.CREATE(NIL, mainHeap, SIZEOF(TClassInfo), numClasses));
001579          XferLeft(Ptr(pClasses), @hMyClasses^^.records, numClasses * SIZEOF(TClassInfo));
```

```
001580
001581          hMySTables := THSTables(TDynamicArray.CREATE(NIL, mainHeap, SIZEOF(TPSliceTable), numClasses));
001582          XferLeft(Ptr(pSTables), @hMySTables^^.records, numClasses * SIZEOF(TPSliceTable));
001583
001584          hMyAuthors := THAuthors(TDynamicArray.CREATE(NIL, mainHeap, SIZEOF(TA32), numAuthors));
001585          XferLeft(Ptr(pAuthors), @hMyAuthors^^.records, numAuthors * SIZEOF(TA32));
001586
001587          hMyAliases := THAliases(TDynamicArray.CREATE(NIL, mainHeap, SIZEOF(TA8), numAliases));
001588          XferLeft(Ptr(pAliases), @hMyAliases^^.records, numAliases * SIZEOF(TA8));
001589  *****)
001590
001591          hMyClasses := THClasses(TArray.CREATE(NIL, mainHeap, numClasses, SIZEOF(TClassInfo)));
001592          TArray(hMyClasses).EditAt(1, numClasses);
001593          XferLeft(Ptr(pClasses), @hMyClasses^^.records, numClasses * SIZEOF(TClassInfo));
001594
001595          hMySTables := THSTables(TArray.CREATE(NIL, mainHeap, numClasses, SIZEOF(TPSliceTable)));
001596          TArray(hMySTables).EditAt(1, numClasses);
001597          XferLeft(Ptr(pSTables), @hMySTables^^.records, numClasses * SIZEOF(TPSliceTable));
001598
001599          hMyAuthors := THAuthors(TArray.CREATE(NIL, mainHeap, numAuthors, SIZEOF(TA32)));
001600          TArray(hMyAuthors).EditAt(1, numAuthors);
001601          XferLeft(Ptr(pAuthors), @hMyAuthors^^.records, numAuthors * SIZEOF(TA32));
001602
001603          hMyAliases := THAliases(TArray.CREATE(NIL, mainHeap, numAliases, SIZEOF(TA8)));
001604          TArray(hMyAliases).EditAt(1, numAliases);
001605          XferLeft(Ptr(pAliases), @hMyAliases^^.records, numAliases * SIZEOF(TA8));
001606
001607          WITH myWorld DO
001608              BEGIN
001609              infRecs := TArray(hMyClasses);  {&&& field names are a bit confusing}
001610              classes := TArray(hMySTables);
001611              authors := TArray(hMyAuthors);
001612              aliases := TArray(hMyAliases);
001613              END;
001614
001615          hMyHashName := MakeIdxArray(numClasses, TRUE);
001616          FOR i := 1 TO numClasses DO
001617              hMyHashName^^.records[LookupName(hMyClasses^^.records[i].classAlpha, TRUE)] := i;
001618          END;
001619  END;
001620
001621
001622  { ================================== METHODS OF CLASSES ================================== }
001623
001624
001625
001626
001627  METHODS OF TObject;
```

```
001628
001629
001630  {$S sStartup}
001631      PROCEDURE TObject.Become(object: TObject);
001632          LABEL   1;
001633
001634          VAR hSelf:  TH;
001635              hObj:   TH;
001636              bkSelf: TBk;
001637              bkObj:  TBk;
001638              p:      TP;
001639  {$IFC LibraryVersion <= 20}
001640              oh:     TC;
001641  {$ELSEC}
001642              tempBP: TBp;
001643  {$ENDC}
001644      BEGIN
001645          {$IFC fTrace}BP(4);{$ENDC}
001646          IF SELF.Heap <> object.Heap THEN
001647              BEGIN
001648              {$IFC fDbgObject}
001649              WriteLn(ORD(SELF));
001650              ABCBreak('Attempt to Become an object on another heap', ORD(object));
001651              {$ENDC}
001652              GOTO 1;
001653              END;
001654
001655          hSelf := TH(SELF);
001656          hObj := TH(object);
001657
001658          bkSelf := TBk(ORD(hSelf^) - 4);
001659          bkObj := TBk(ORD(hObj^) - 4);
001660
001661          p := hSelf^;
001662          hSelf^ := hObj^;
001663          hObj^ := p;
001664
001665  {$IFC LibraryVersion <= 20}
001666          oh := bkSelf^.oh;
001667          bkSelf^.oh := bkObj^.oh;
001668          bkObj^.oh := oh;
001669  {$ELSEC}
001670          tempBP := bkSelf^.bp;
001671          bkSelf^.bp := bkObj^.bp;
001672          bkObj^.bp := tempBP;
001673  {$ENDC}
001674
001675          object.Free;
```

```
001676  1:
001677          {$IFC fTrace}EP;{$ENDC}
001678      END;
001679
001680
001681  {$S sStartup}
001682      FUNCTION  TObject.Class: TClass;
001683      BEGIN
001684          {$IFC fMaxTrace}BP(1);{$ENDC}
001685          Class := ClassPtr(Handle(SELF));
001686          {$IFC fMaxTrace}EP;{$ENDC}
001687      END;
001688
001689
001690  {$S sRes}
001691      FUNCTION  TObject.Clone(heap: THeap): TObject;
001692      BEGIN
001693          {$IFC fMaxTrace}BP(1);{$ENDC}
001694          Clone := SELF.CloneObject(heap);
001695          {$IFC fMaxTrace}EP;{$ENDC}
001696      END;
001697
001698
001699  {$S sRes}
001700      FUNCTION  TObject.CloneObject(heap: THeap): TObject;
001701          VAR hz:     THz;
001702              size:   INTEGER;
001703              source: TH;
001704              dest:   TH;
001705      BEGIN
001706          {$IFC fTrace}BP(2);{$ENDC}
001707          hz := THz(heap);
001708          source := TH(SELF);
001709          size := cbDataOfH(hz, source);
001710          dest := HAllocate(hz, size);
001711          XferLeft(@source^^, @dest^^, size);
001712          CloneObject := TObject(dest);
001713          {$IFC fTrace}EP;{$ENDC}
001714      END;
001715
001716
001717  {$S sStartup}
001718      PROCEDURE TObject.Free;
001719      BEGIN
001720          {$IFC fTrace}BP(3);{$ENDC}
001721          SELF.FreeObject;
001722          {$IFC fTrace}EP;{$ENDC}
001723      END;
```

```
001724
001725
001726  {$S sStartup}
001727      PROCEDURE TObject.FreeObject;
001728          VAR heap:       THeap;
001729              numObjects: INTEGER;
001730      BEGIN
001731          {$IFC fTrace}BP(4);{$ENDC}
001732          heap := SELF.Heap;
001733          FreeH(THz(heap), TH(SELF));
001734          {$IFC fTrace}EP;{$ENDC}
001735      END;
001736
001737
001738  {$S sStartup}
001739      FUNCTION  TObject.Heap: THeap;
001740      BEGIN
001741          {$IFC fMaxTrace}BP(1);{$ENDC}
001742          {$IFC fMaxTrace}EP;{$ENDC}
001743          Heap := THeap(HzFromH(TH(SELF)));
001744      END;
001745
001746
001747  {$S sRes}
001748      FUNCTION  TObject.HeapBytes: INTEGER;
001749      BEGIN
001750          {$IFC fTrace}BP(2);{$ENDC}
001751          HeapBytes := CbDataOfH(HzFromH(TH(SELF)), TH(SELF));
001752          {$IFC fTrace}EP;{$ENDC}
001753      END;
001754
001755
001756  {$S sLOX}
001757      PROCEDURE TObject.Read(s: TStringScanner);
001758      BEGIN
001759          {$IFC fTrace}BP(2);{$ENDC}
001760          s.XferFields(xRead, SELF);
001761          {$IFC fTrace}EP;{$ENDC}
001762      END;
001763
001764
001765  {$S sLOX}
001766      PROCEDURE TObject.Write(s: TStringScanner);
001767      BEGIN
001768          {$IFC fTrace}BP(2);{$ENDC}
001769          s.XferFields(xWrite, SELF);
001770          {$IFC fTrace}EP;{$ENDC}
001771      END;
```

```
001772
001773
001774      {$IFC fDebugMethods}
001775      {$S SgCLAdbg}
001776      PROCEDURE TObject.Debug(numLevels: INTEGER; memberTypeStr: S255);
001777          VAR class:      TClass;
001778              name:       TClassName;
001779              str:        S255;
001780              {$IFC fTrace}
001781              oldFlag:    BOOLEAN;
001782              {$ENDC}
001783
001784          PROCEDURE SupplyObjFields(PROCEDURE Field(nameAndType: S255));
001785          BEGIN
001786              SELF.Fields(Field);
001787          END;
001788
001789      BEGIN
001790          {$IFC fTrace}
001791          oldFlag  := fDebugRecursion;
001792          fDebugRecursion := TRUE;
001793          {$ENDC}
001794
001795          class := SELF.Class;
001796          CpToCn(TPSliceTable(class), TS8(name));
001797          TrimBlanks(@name);
001798          WrStr(Concat(name, ' '));
001799
001800          {$IFC fDebugMethods}
001801          IF numLevels > 0 THEN
001802              WriteDRecord(numLevels, Handle(SELF), 4, SupplyObjFields);  {4 skips method table ptr}
001803          {$ELSEC}
001804          LIntToHex(ORD(SELF), @str);
001805          str := Concat('-- $', str);
001806          IF NOT ValidObject(Handle(SELF)) THEN
001807              str := Concat('Invalid Object', str);
001808          WrStr(str);
001809          {$ENDC}
001810
001811          {$IFC fTrace}
001812          fDebugRecursion := oldFlag;
001813          {$ENDC}
001814      END;
001815      {$S SgCLAres}
001816      {$ENDC}
001817
001818
001819      {$IFC fDebugMethods}
```

```
001820      {$S SgCLAdbg}
001821      PROCEDURE TObject.Fields(PROCEDURE Field(nameAndType: S255));
001822      BEGIN
001823      END;
001824      {$S SgCLAres}
001825      {$ENDC}
001826
001827
001828      {$S SgCLAcld}
001829      PROCEDURE TObject.Convert(fromVersion: Byte);
001830      BEGIN
001831          {$IFC fMaxTrace}BP(1);{$ENDC}
001832          {$IFC fMaxTrace}EP;{$ENDC}
001833      END;
001834      {$S SgCLAres}
001835
001836
001837      {$S SgCLAcld}
001838      FUNCTION  TObject.JoinClass(newClass: TClass): TObject;
001839          VAR oldClass:   TClass;
001840      BEGIN
001841          {$IFC fTrace}BP(7);{$ENDC}
001842          oldClass := SELF.Class;
001843          IF NOT ClassDescendsFrom(oldClass, newClass) THEN
001844              IF ClassDescendsFrom(newClass, oldClass) THEN
001845                  Recreate(SELF, SizeOfCp(TPSliceTable(oldClass)),
001846                                 SizeOfCP(TPSliceTable(newClass)), TPSliceTable(newClass)) (*^*)
001847              ELSE
001848                  {$IFC fDbgObject}
001849                  ABCBreak('An Object cannot move to an unrelated class', ORD(newClass))
001850                  {$ENDC};
001851          JoinClass := SELF;
001852          {$IFC fTrace}EP;{$ENDC}
001853      END;
001854      {$S SgCLAres}
001855
001856
001857  {$S sInit1}
001858  BEGIN {Class Initialization}
001859
001860      InitClascal(ClascalError);   {Provide an error routine in case of errors in Clascal run-time support}
001861      InitObject;                  {Do remaining initialization}
001862
001863      UnitAuthor('Apple');
001864
001865      cObject := THISCLASS;
001866
001867  END;
```

```
001868
001869
001870
001871

End of File -- Lines: 1871 Characters: 57741
```

```
================================================================================================
FILE: "LIBTK/UOBJECT3.TEXT"
================================================================================================

000001  {INCLUDE FILE UOBJECT3 -- COLLECTIONS}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004  {Segments: SgCLAini(tialize and Terminate), SgCLAres(ident), SgCLAc(o)ld, SgCLAdbg}
000005
000006  {$S sResDat}
000007  PROCEDURE XferContiguous(whichWay: xReadWrite; collection: TCollection; alsoSkip: INTEGER; s: TStringScanner);
000008          {Transfer the size (as an INTEGER), class-specific fields (after alsoSkip bytes), and all members.
000009           Do not recur on the members.
000010           Do not transfer the class, the dynStart (=SizeOfClass), or the hole info (=zero).
000011           When reading, append the elements that are read.
000012           This only works for contiguous objects up to 32K members in size.}
000013      VAR size:       INTEGER;
000014          numToXfer:  INTEGER;
000015  BEGIN
000016      {$IFC fTrace}BP(3);{$ENDC}
000017      size := collection.size;
000018      collection.StopEdit;
000019      CASE whichWay OF
000020          xRead:
000021              BEGIN
000022              numToXfer := s.ReadNumber(2);
000023              collection.EditAt(size + 1, numToXfer);
000024              size := collection.size;
000025              END;
000026          xWrite:
000027              BEGIN
000028              numToXfer := size;
000029              s.WriteNumber(numToXfer, 2);
000030              END;
000031          END;
000032      s.XferSequential(whichWay,
000033                       Ptr(ORD(Handle(collection)^) + SIZEOF(TCollection) + alsoSkip),
000034                       size * collection.MemberBytes);
000035      {$IFC fTrace}EP;{$ENDC}
000036  END;
000037
000038
000039  {INVARIANT ON TCollections:
000040      given a collection c,
000041   the elements of the collection are stored at physical indices:
000042          [1..c.holeStart] and [c.holeStart+c.holeSize+1..c.Size+c.holeSize]
000043   the hole occupies physical indices:
```

```
000044          [c.holeStart+1..c.holeStart+c.holeSize]
000045  }
000046
000047  METHODS OF TCollection;
000048
000049
000050  {$S sResDat}
000051      FUNCTION  TCollection.CREATE(object: TObject; heap: THeap; initialSlack: INTEGER): TCollection;
000052      BEGIN
000053          {$IFC fTrace}BP(1);{$ENDC}
000054          IF object = NIL THEN
000055              ABCBreak('TCollection.CREATE must be passed an already-allocated object by a subclass CREATE', 0);
000056          SELF := TCollection(object);
000057          WITH SELF DO
000058              BEGIN
000059              size := 0;
000060        {$H-} dynStart := SizeOfClass(SELF.Class); {$H+}
000061              holeStart := 0;
000062              holeSize := initialSlack;
000063              holeStd := 0;
000064              END;
000065          {$IFC fTrace}EP;{$ENDC}
000066      END;
000067
000068
000069  {$S sResDat}
000070      FUNCTION  TCollection.Clone(heap: THeap): TObject;
000071          VAR numMembers: INTEGER;
000072              collection: TCollection;
000073      BEGIN
000074          {$IFC fTrace}BP(4);{$ENDC}
000075          numMembers := SELF.size;
000076          collection := TCollection(NewDynObject(heap, SELF.Class, numMembers * SELF.MemberBytes));
000077          XferLeft(Ptr(Handle(SELF)^), Ptr(Handle(collection)^), SELF.dynStart);
000078          collection := TCollection.CREATE(collection, heap, numMembers);
000079          collection.InsManyAt(1, SELF, 1, numMembers);
000080          Clone := collection;
000081          {$IFC fTrace}EP;{$ENDC}
000082      END;
000083
000084
000085      {$IFC fDebugMethods}
000086      {$S SgCLAdbg}
000087      PROCEDURE TCollection.Fields(PROCEDURE Field(nameAndType: S255));
000088      BEGIN
000089          SUPERSELF.Fields(Field);
000090          Field('size: LONGINT');
000091          Field('dynStart: INTEGER');
```

```
000092          Field('holeStart: INTEGER');
000093          Field('holeSize: INTEGER');
000094          Field('holeStd: INTEGER');
000095      END;
000096      {$S SgCLAres}
000097      {$ENDC}
000098
000099
000100      {$IFC fCheckIndices}
000101      {$S SgCLAdbg}
000102      PROCEDURE TCollection.CheckIndex(index: LONGINT);
000103      BEGIN
000104          {$IFC fMaxTrace}BP(1);{$ENDC}
000105          {$IFC fMaxTrace}EP;{$ENDC}
000106          IF (index < 1) OR (index > SELF.size) THEN
000107              ABCBreak('CheckIndex', index);
000108      END;
000109      {$S SgCLAres}
000110      {$ENDC}
000111
000112
000113  {$S sResDat}
000114      FUNCTION  TCollection.AddrMember(i: LONGINT): LONGINT;
000115      BEGIN
000116          {$IFC fTrace}BP(3);{$ENDC}
000117          {$IFC fDbgObject}
000118          IF SELF.dynStart = MAXINT THEN
000119              ABCBreak('No dynamic part', i);
000120          {$ENDC}
000121          {$IFC fCheckIndices}
000122          IF fCheckIndices THEN
000123              IF (i < 1) OR (i > SELF.size+1) THEN
000124                  ABCBreak('CheckIndex', i);
000125          {$ENDC}
000126
000127          IF i > SELF.holeStart THEN
000128              i := i + SELF.holeSize;
000129          {i is now a physical index}
000130
000131          AddrMember := TpLONGINT(SELF)^ + SELF.dynStart + (SELF.MemberBytes * (i - 1));
000132          {$IFC fTrace}EP;{$ENDC}
000133      END;
000134
000135
000136  {$S sResDat}
000137      PROCEDURE TCollection.CopyMembers(dstAddr, startIndex, howMany: LONGINT);
000138          VAR memberBytes:    INTEGER;
000139              beforeHole:     INTEGER;
```

```
000140              srcAddr:        LONGINT;
000141              j:              INTEGER;
000142              offset:         INTEGER;
000143              numBytes:       INTEGER;
000144      BEGIN
000145          {$IFC fTrace}BP(3);{$ENDC}
000146          IF startIndex < 1 THEN
000147              startIndex := 1;
000148          howMany := Min(howMany, SELF.size + 1 - startIndex);
000149
000150          IF (howMany > 0) AND (startIndex <= SELF.size) THEN
000151              BEGIN
000152              memberBytes := SELF.MemberBytes;
000153
000154              beforeHole := Min(howMany, SELF.holeStart + 1 - startIndex);
000155              srcAddr := SELF.AddrMember(startIndex);
000156
000157              IF beforeHole > 0 THEN
000158                  BEGIN
000159                  numBytes := beforeHole * memberBytes;
000160
000161                  XferLeft(Ptr(srcAddr), Ptr(dstAddr), numBytes);
000162                  dstAddr := dstAddr + numBytes;
000163                  END
000164              ELSE
000165                  beforeHole := 0;
000166
000167              IF beforeHole < howMany THEN
000168                  BEGIN
000169                  srcAddr := SELF.AddrMember(startIndex + beforeHole);
000170                  XferLeft(Ptr(srcAddr), Ptr(dstAddr), (howMany - beforeHole) * memberBytes);
000171                  END;
000172              END;
000173          {$IFC fTrace}EP;{$ENDC}
000174      END;
000175
000176
000177  {$S sResDat}
000178  {AFTER EXECUTING THIS METHOD:
000179      IF deltaMembers >= 0,
000180          physical positions [atIndex..atIndex+deltaMembers-1] are available for adding new members.
000181
000182      IF deltaMembers < 0,
000183          actual members [atIndex..atIndex-deltaMembers+1] have been removed.
000184
000185  NOTE: This routine does not preserve the TCollection invariant.
000186  }
000187      PROCEDURE TCollection.EditAt(atIndex: LONGINT; deltaMembers: INTEGER);
```

```
000188          VAR oldHoSize:      INTEGER;
000189              newHoSize:      INTEGER;
000190              oldHoStart:     INTEGER;
000191              newHoStart:     INTEGER;
000192              maxHoStart:     INTEGER;
000193              minHoStart:     INTEGER;
000194              size:           INTEGER;
000195              b:              0..1;
000196      BEGIN                                           {Removes any hole it creates unless holdStd <> 0}
000197          {$IFC fTrace}BP(4);{$ENDC}
000198          {$IFC fDbgObject}
000199          IF SELF.dynStart = MAXINT THEN
000200              ABCBreak('No dynamic part', atIndex);
000201          {$ENDC}
000202
000203          {Force atIndex and deltaMembers into the valid range}
000204          atIndex := Max(1, Min(atIndex, SELF.size + 1));
000205
000206          IF deltaMembers < 0 THEN
000207              deltaMembers := Min(0, Max(deltaMembers, atIndex - SELF.size - 1));
000208
000209  (***** Range checks not necessary with the above code
000210          {$IFC fCheckIndices}
000211          IF fCheckIndices THEN
000212              BEGIN
000213              IF atIndex <> (SELF.size + 1) THEN
000214                  SELF.CheckIndex(atIndex);
000215              IF deltaMembers < 0 THEN
000216                  SELF.CheckIndex(atIndex - 1 - deltaMembers);
000217              END;
000218          {$ENDC}
000219  *****)
000220
000221          oldHoSize := SELF.holeSize;
000222          oldHoStart := SELF.holeStart;
000223
000224          IF (deltaMembers < 0) AND ((oldHoStart + 1) = atIndex) THEN {the hole is right before the deletion}
000225              SELF.holeStart := oldHoStart - deltaMembers {deltaMembers is going to be added in again later}
000226          ELSE
000227              BEGIN
000228              newHoStart := atIndex - 1 - Min(deltaMembers, 0);
000229              IF (deltaMembers > oldHoSize) OR (newHoStart <> oldHoStart) THEN
000230                  BEGIN
000231                  maxHoStart := Max(oldHoStart, newHoStart);
000232                  newHoSize := Max(oldHoSize, deltaMembers);
000233
000234                  IF newHoSize > oldHoSize THEN
000235                      BEGIN
```

```
000236                  {increase the space allocated to the collection, and shift the collection so that the
000237                   the last real element is at the end of the space allocated to the collection;
000238                   but only move REAL elements that will end up after the hole}
000239
000240                  size := SELF.size;
000241                  newHoSize := Max(newHoSize, SELF.holeStd);
000242                  SELF.ResizeColl(size + newHoSize);
000243                  SELF.ShiftColl(maxHoStart + oldHoSize, maxHoStart + newHoSize, size - maxHoStart);
000244
000245                  {Explanation of the above line:
000246                      maxHoStart = max # real elements before the hole (in initial and final collections)
000247                      size = # real elements in the initial collection
000248                      therefore, size - maxHoStart is min # real elements after the hole, which
000249                          is the right number of elements to move
000250
000251                      the allocated size of the collection is size + newHoSize (from SELF.ResizeColl)
000252                      to get the last real element we are moving to be at the end of the allocated space,
000253                          we need to move the first element to
000254                          allocated size of collection - # elements moving
000255                      =   size + newHoSize - (size - maxHoStart)
000256                      =   maxHoStart + newHoSize
000257
000258                      we increased the size of the collection by newHoSize - oldHoSize
000259                      therefore the first source element must be
000260                          first destination element - (newHoSize - oldHoSize)
000261                      =   maxHoStart + newHoSize - (newHoSize - oldHoSize)
000262                      =   maxHoStart + oldHoSize
000263                  }
000264                  END;
000265
000266              IF newHoStart <> oldHoStart THEN
000267                  BEGIN
000268                  b := ORD(newHoStart > oldHoStart);    {1 if hole is moving right and data is moving left,
000269                                                         0 otherwise}
000270                  minHoStart := Min(oldHoStart, newHoStart);
000271                  SELF.ShiftColl(minHoStart + oldHoSize*b, minHoStart + newHoSize*(1-b),
000272                                  maxHoStart - minHoStart);
000273                  END;
000274
000275              SELF.holeStart := newHoStart;
000276              SELF.holeSize := newHoSize;
000277              END;
000278          END;
000279
000280      WITH SELF DO
000281          BEGIN
000282          size := size + deltaMembers;
000283          holeSize := holeSize - deltaMembers;
```

```
000284              holeStart := holeStart + deltaMembers;
000285              IF oldHoSize = 0 THEN
000286                  IF holeStd = 0 THEN
000287                      IF holeSize > 0 THEN
000288                  {$H-}   SELF.StopEdit;  {$H+}
000289              END;
000290          {$IFC fTrace}EP;{$ENDC}
000291      END;
000292
000293
000294  {$S SgABCdat}
000295      FUNCTION  TCollection.Equals(otherCollection: TCollection): BOOLEAN;
000296          LABEL 1;
000297          VAR memberBytes: INTEGER;
000298              size:        INTEGER;
000299              i:           INTEGER;
000300      BEGIN
000301          {$IFC fTrace}BP(3);{$ENDC}
000302          Equals := FALSE;
000303          memberBytes := SELF.MemberBytes;
000304          size := SELF.size;
000305          IF SELF = otherCollection THEN
000306              Equals := TRUE
000307          ELSE
000308          IF size = otherCollection.size THEN
000309              IF memberBytes = otherCollection.MemberBytes THEN
000310                  BEGIN
000311                  FOR i := 1 TO size DO
000312                      IF NOT EqualBytes(Ptr(SELF.AddrMember(i)), Ptr(otherCollection.AddrMember(i)),
000313                                                              memberBytes) THEN
000314                              GOTO 1;
000315                  Equals := TRUE;
000316                  END;
000317          1:
000318          {$IFC fTrace}EP;{$ENDC}
000319      END;
000320
000321
000322  {$S sResDat}
000323      PROCEDURE TCollection.InsManyAt(i: LONGINT; otherCollection: TCollection; index, howMany: LONGINT);
000324      BEGIN                                                   {Stops edit if it wasn't explicitly started}
000325          {$IFC fTrace}BP(3);{$ENDC}
000326          {$IFC fDbgObject}
000327          IF SELF.dynStart = MAXINT THEN
000328              ABCBreak('No dynamic part', i);
000329          {$ENDC}
000330
000331          {$IFC fCheckIndices}
```

```
000332          IF fCheckIndices AND (howMany > 0) THEN
000333              BEGIN {i is checked by EditAt}
000334              IF SELF.memberBytes <> otherCollection.MemberBytes THEN
000335                  BEGIN
000336                  WriteLn;
000337                  WriteLn('*** ERROR: Tried to insert ', otherCollection.MemberBytes:1,
000338                          '-byte Members into a TCollection with ', SELF.memberBytes, '-byte Members');
000339                  ABCbreak('InsManyAt', howMany);
000340                  END;
000341  (***** Dont need range checks anymore
000342              otherCollection.CheckIndex(index);
000343              otherCollection.CheckIndex(index + howMany - 1);
000344  *****)
000345              END;
000346          {$ENDC}
000347
000348          IF howMany > 0 THEN
000349              BEGIN
000350              SELF.EditAt(i, howMany);
000351
000352              otherCollection.CopyMembers(SELF.AddrMember(i), index, howMany);
000353              END;
000354          {$IFC fTrace}EP;{$ENDC}
000355      END;
000356
000357
000358  {$S SgABCdat}
000359      PROCEDURE TCollection.InsNullsAt(i, howMany: LONGINT);
000360          VAR dstAddr:        LONGINT;
000361      BEGIN
000362          {$IFC fTrace}BP(4);{$ENDC}
000363          {$IFC fDbgObject}
000364          IF SELF.dynStart = MAXINT THEN
000365              ABCBreak('No dynamic part', i);
000366          {$ENDC}
000367
000368          SELF.EditAt(i, howMany);
000369          IF howMany > 0 THEN
000370              BEGIN
000371              dstAddr := SELF.AddrMember(i);
000372              TPByte(dstAddr)^ := 0;
000373              XferLeft(Ptr(dstAddr), Ptr(dstAddr + 1), howMany * SELF.MemberBytes-1);
000374                  {WARNING: The success of the preceding line depends on the fact the XferLeft
000375                          copies data 1 byte at a time; use of a routine that tries to optimize the
000376                          transfer will negatively impact the correctness of this method.}
000377              END;
000378          {$IFC fTrace}EP;{$ENDC}
000379      END;
```

```
000380
000381
000382  {$S sResDat}
000383  {NOTE: This routine does not preserve the TCollection invariant.}
000384
000385      PROCEDURE TCollection.ResizeColl(membersPlusHole: INTEGER);
000386      BEGIN
000387          {$IFC fTrace}BP(4);{$ENDC}
000388          {$IFC fDbgObject}
000389          IF SELF.dynStart = MAXINT THEN
000390              ABCBreak('No dynamic part', membersPlusHole);
000391          {$ENDC}
000392
000393          IF membersPlusHole <> (SELF.size + SELF.holeSize) THEN
000394              ResizeDynObject(SELF, SELF.dynStart + (membersPlusHole * SELF.MemberBytes));
000395          {$IFC fTrace}EP;{$ENDC}
000396      END;
000397
000398
000399  {$S sResDat}
000400  {NOTE: This routine does not preserve the TCollection invariant.}
000401
000402      PROCEDURE TCollection.ShiftColl(afterSrcIndex, afterDstIndex, howMany: INTEGER);
000403          VAR memberBytes:    INTEGER;
000404              numBytes:       INTEGER;
000405              startAddr:      LONGINT;
000406              srcAddr:        LONGINT;
000407              dstAddr:        LONGINT;
000408      BEGIN
000409          {$IFC fTrace}BP(4);{$ENDC}
000410          {$IFC fDbgObject}
000411          IF SELF.dynStart = MAXINT THEN
000412              ABCBreak('No dynamic part', howMany);
000413          {$ENDC}
000414
000415          IF (howMany > 0) AND (afterSrcIndex <> afterDstIndex) THEN
000416              BEGIN
000417              memberBytes := SELF.MemberBytes;
000418              numBytes := howMany * memberBytes;
000419
000420              startAddr := TpLONGINT(SELF)^ + SELF.dynStart;
000421              srcAddr := startAddr + afterSrcIndex * memberBytes;
000422              dstAddr := startAddr + afterDstIndex * memberBytes;
000423
000424              IF afterSrcIndex < afterDstIndex THEN
000425                  XferRight(Ptr(srcAddr), Ptr(dstAddr), numBytes)
000426              ELSE
000427                  XferLeft(Ptr(srcAddr), Ptr(dstAddr), numBytes);
```

```
000428              END;
000429          {$IFC fTrace}EP;{$ENDC}
000430      END;
000431
000432
000433  {$S SgABCdat}
000434      PROCEDURE TCollection.StartEdit(withSlack: INTEGER);
000435      BEGIN
000436          {$IFC fTrace}BP(4);{$ENDC}
000437          {$IFC fDbgObject}
000438          IF SELF.dynStart = MAXINT THEN
000439              ABCBreak('No dynamic part', withSlack);
000440          {$ENDC}
000441
000442          SELF.holeStd := withSlack;
000443          {$IFC fTrace}EP;{$ENDC}
000444      END;
000445
000446
000447  {$S sResDat}
000448      PROCEDURE TCollection.StopEdit;
000449      BEGIN
000450          {$IFC fTrace}BP(4);{$ENDC}
000451          {$IFC fDbgObject}
000452          IF SELF.dynStart = MAXINT THEN
000453              ABCBreak('No dynamic part', 0);
000454          {$ENDC}
000455
000456          IF SELF.holeStart < SELF.size THEN
000457              SELF.EditAt(SELF.size + 1, 0);
000458          SELF.ResizeColl(SELF.size);
000459          SELF.holeStd := 0;
000460          SELF.holeSize := 0;
000461          {$IFC fTrace}EP;{$ENDC}
000462      END;
000463
000464
000465  {$S sInit1}
000466  BEGIN
000467
000468      {$IFC fCheckIndices}
000469      fCheckIndices := FALSE;
000470      {$ENDC}
000471
000472  END;
000473  {$S SgCLAres}
000474
000475
```

```
000476  METHODS OF TList;
000477
000478
000479  {$S sResDat}
000480      FUNCTION  TList.CREATE(object: TObject; heap: THeap; initialSlack: INTEGER): TList;
000481      BEGIN
000482          {$IFC fTrace}BP(1);{$ENDC}
000483          IF object = NIL THEN
000484              object := NewDynObject(heap, THISCLASS, initialSlack * SIZEOF(Handle));
000485          SELF := TList(TCollection.CREATE(object, heap, initialSlack));
000486          {$IFC fTrace}EP;{$ENDC}
000487      END;
000488
000489
000490  {$S sResDat}
000491      PROCEDURE TList.Free;
000492      BEGIN
000493          {$IFC fTrace}BP(4);{$ENDC}
000494          SELF.Each(Free);
000495          SUPERSELF.Free;
000496          {$IFC fTrace}EP;{$ENDC}
000497      END;
000498
000499
000500  {$S SgABCdat}
000501      FUNCTION  TList.Clone(heap: THeap): TObject;
000502          VAR l:  TList;
000503              j:  INTEGER;
000504              x:  TObject;
000505      BEGIN
000506          {$IFC fTrace}BP(4);{$ENDC}
000507          l := TList(SUPERSELF.Clone(heap));
000508          FOR j := 1 TO l.size DO
000509              BEGIN
000510              x := SELF.At(j);
000511              IF x <> NIL THEN
000512                  l.PutAt(j, x.Clone(heap), FALSE);
000513              END;
000514          Clone := l;
000515          {$IFC fTrace}EP;{$ENDC}
000516      END;
000517
000518
000519      {$IFC fDebugMethods}
000520      {$S SgCLAdbg}
000521      PROCEDURE TList.Debug(numLevels: INTEGER; memberTypeStr: S255);
000522      VAR s:          TListScanner;
000523          obj:        TObject;
```

```
000524          str:        S8;
000525          first:      BOOLEAN;
000526          {$IFC fTrace}
000527          oldFlag:    BOOLEAN;
000528          {$ENDC}
000529      BEGIN
000530          {$IFC fTrace}
000531          oldFlag := fDebugRecursion;
000532          fDebugRecursion := TRUE;
000533          {$ENDC}
000534
000535          SUPERSELF.Debug(numLevels, '');         { this prints other fields of the list }
000536          IF numLevels > 0 THEN
000537              BEGIN
000538              WrStr('(');
000539              IF numLevels = 1 THEN           { compressed list of classes }
000540                  SELF.DebugMembers
000541              ELSE                            { list of classes and their handles }
000542                  BEGIN
000543                  s := SELF.Scanner;
000544                  IF s.position = SELF.holeStart THEN
000545                      Write('<=HOLE=>');
000546                  first := TRUE;
000547                  WHILE s.Scan(obj) DO
000548                      BEGIN
000549                      IF NOT first THEN
000550                          WrStr(', ');
000551                      first := FALSE;
000552                      IF obj = NIL THEN
000553                          WrStr('NIL')
000554                      ELSE IF ValidObject(Handle(obj)) THEN
000555                          obj.Debug(numLevels-2, '')
000556                      ELSE
000557                          WrStr('<Invalid Object>');
000558                      IF numLevels = 2 THEN
000559                          BEGIN
000560                          LIntToHex(ORD4(obj), @str);
000561                          WrStr(CONCAT(': $', str));
000562                          END;
000563                      IF s.position = SELF.holeStart THEN
000564                          Write('<=HOLE=>');
000565                      END;
000566                  END;
000567              WrStr(')');
000568              END;
000569
000570          {$IFC fTrace}
000571          fDebugRecursion := oldFlag;
```

```
000572          {$ENDC}
000573      END;
000574      {$S SgCLAres}
000575
000576
000577      {$S SgCLAdbg}
000578      PROCEDURE TList.DebugMembers;
000579          VAR y:              TObject;
000580              s:              TListScanner;
000581              str:            S8;
000582              initial:        BOOLEAN;
000583              class:          TClass;
000584              thisClass:      TClassName;
000585              prevClass:      TClassName;
000586              sameClass:      INTEGER;
000587              charCount:      INTEGER;
000588
000589          PROCEDURE WriteMembers;
000590              VAR charsNeeded:    INTEGER;
000591          BEGIN
000592              IF sameClass = 0 THEN EXIT(WriteMembers);
000593              IF sameClass = 1 THEN
000594                  charsNeeded := 10
000595              ELSE
000596                  charsNeeded := 13;
000597              IF initial THEN
000598                  initial := FALSE
000599              ELSE IF (charCount + charsNeeded) > 70 THEN
000600                  BEGIN
000601                  WrStr(',');
000602                  WrLn;
000603                  WrStr('          ');
000604                  charCount := 10;
000605                  END
000606              ELSE
000607                  WrStr(', ');
000608
000609              str := prevClass;
000610              WrStr(str);
000611
000612              IF sameClass > 1 THEN
000613                  BEGIN
000614                  IntToStr(sameClass, @str);
000615                  WrStr(CONCAT('*', str));
000616                  END;
000617
000618              charCount := charCount + charsNeeded;
000619          END;
```

```
000620
000621      BEGIN
000622          IF SELF.size > 0 THEN {prevent initialization anomaly in BP(i)/EP}
000623              BEGIN
000624              charCount := cMin(indentTrace, 20) + 30;
000625              initial := TRUE;
000626              sameClass := 0;
000627              prevClass := '';
000628              s := SELF.Scanner;
000629              WHILE s.Scan(y) DO
000630                  BEGIN
000631                  IF y = NIL THEN
000632                      thisClass := 'NIL'
000633                  ELSE IF ValidObject(Handle(y)) THEN
000634                      BEGIN
000635                      class := y.Class;
000636                      CpToCn(TPSliceTable(class), TS8(thisClass));
000637                      END
000638                  ELSE
000639                      thisClass := '????????';
000640
000641                  IF thisClass <> prevClass THEN
000642                      BEGIN
000643                      WriteMembers;
000644                      sameClass := 1;
000645                      END
000646                  ELSE
000647                      sameClass := sameClass + 1;
000648
000649                  prevClass := thisClass;
000650                  END;
000651              WriteMembers;
000652              END;
000653      END;
000654      {$S SgCLAres}
000655      {$ENDC}
000656
000657
000658  {$S sResDat}
000659      FUNCTION  TList.At(i: LONGINT): TObject;
000660      BEGIN
000661          {$IFC fTrace}BP(3);{$ENDC}
000662          {$IFC fCheckIndices}
000663          IF fCheckIndices THEN
000664              SELF.CheckIndex(i);
000665          {$ENDC}
000666
000667         {At := TPObject(SELF.AddrMember(i))^;  but for speed...}
```

```
000668
000669          IF i > SELF.holeStart THEN
000670              i := i + SELF.holeSize;
000671          At := TPObject(TpLONGINT(SELF)^ + SELF.dynStart + (4 * (i - 1)))^;
000672          {$IFC fTrace}EP;{$ENDC}
000673      END;
000674
000675
000676  {$S SgABCdat}
000677      PROCEDURE TList.DelAll(freeOld: BOOLEAN);
000678      BEGIN
000679          {$IFC fTrace}BP(4);{$ENDC}
000680          IF freeOld THEN
000681              SELF.Each(Free);
000682          SELF.EditAt(1, -SELF.size);
000683          {$IFC fTrace}EP;{$ENDC}
000684      END;
000685
000686
000687  {$S SgABCdat}
000688      PROCEDURE TList.DelAt(i: LONGINT; freeOld: BOOLEAN);
000689      BEGIN
000690          {$IFC fTrace}BP(4);{$ENDC}
000691          {$IFC fCheckIndices}
000692          IF fCheckIndices THEN
000693              SELF.CheckIndex(i);
000694          {$ENDC}
000695
000696          IF freeOld THEN
000697              Free(SELF.At(i));
000698          SELF.EditAt(i, -1);
000699          {$IFC fTrace}EP;{$ENDC}
000700      END;
000701
000702
000703  {$S SgABCdat}
000704      PROCEDURE TList.DelFirst(freeOld: BOOLEAN);
000705      BEGIN
000706          {$IFC fTrace}BP(3);{$ENDC}
000707          SELF.DelAt(1, freeOld);
000708          {$IFC fTrace}EP;{$ENDC}
000709      END;
000710
000711
000712  {$S SgABCdat}
000713      PROCEDURE TList.DelLast(freeOld: BOOLEAN);
000714      BEGIN
000715          {$IFC fTrace}BP(3);{$ENDC}
```

```
000716          SELF.DelAt(SELF.size, freeOld);
000717          {$IFC fTrace}EP;{$ENDC}
000718      END;
000719
000720
000721  {$S SgABCdat}
000722      PROCEDURE TList.DelManyAt(i, howMany: LONGINT; freeOld: BOOLEAN);
000723          VAR j: INTEGER;
000724      BEGIN
000725          {$IFC fTrace}BP(4);{$ENDC}
000726          IF howMany > 0 THEN
000727              BEGIN
000728              {$IFC fCheckIndices}
000729              IF fCheckIndices THEN
000730                  BEGIN
000731                  SELF.CheckIndex(i);
000732                  SELF.CheckIndex(i+howMany-1);
000733                  END;
000734              {$ENDC}
000735              IF freeOld THEN
000736                  FOR j := 0 TO howMany - 1 DO
000737                      Free(SELF.At(i + j));
000738              SELF.EditAt(i, -howMany);
000739              END;
000740          {$IFC fTrace}EP;{$ENDC}
000741      END;
000742
000743
000744  {$S SgABCdat}
000745      PROCEDURE TList.DelObject(object: TObject; freeOld: BOOLEAN);
000746          VAR y:  TObject;
000747              s:  TListScanner;
000748      BEGIN   {If there is more than one occurrence, and editing is off, this calls StopEdit more than once}
000749          {$IFC fTrace}BP(4);{$ENDC}
000750          s := SELF.Scanner;
000751          WHILE s.Scan(y) DO
000752              IF y = object THEN
000753                  s.Delete(FALSE);
000754          IF freeOld THEN
000755              Free(object);
000756          {$IFC fTrace}EP;{$ENDC}
000757      END;
000758
000759
000760  {$S sResDat}
000761      PROCEDURE TList.Each(PROCEDURE DoToObject(object: TObject));
000762          VAR holeStart:  INTEGER;
000763              offset:     INTEGER;
```

```
000764              j:          INTEGER;
000765              pObject:    TPObject;
000766      BEGIN
000767          {$IFC fTrace}BP(4);{$ENDC}
000768          holeStart := SELF.holeStart;
000769          offset := SELF.dynStart;
000770          FOR j := 0 TO SELF.size - 1 DO
000771              BEGIN
000772              IF j = holeStart THEN
000773                  offset := offset + 4 * SELF.holeSize;
000774              pObject := TPObject(TpLONGINT(SELF)^ + offset);
000775              DoToObject(pObject^);
000776              offset := offset + 4;
000777              END;
000778          {$IFC fTrace}EP;{$ENDC}
000779      END;
000780
000781
000782  {$S sResDat}
000783      FUNCTION  TList.First: TObject;
000784      BEGIN
000785          {$IFC fTrace}BP(3);{$ENDC}
000786          First := SELF.At(1);
000787          {$IFC fTrace}EP;{$ENDC}
000788      END;
000789
000790
000791  {$S sResDat}
000792      PROCEDURE TList.InsAt(i: LONGINT; object: TObject);
000793          VAR pObject:    TPObject;
000794      BEGIN
000795          {$IFC fTrace}BP(4);{$ENDC}
000796          SELF.EditAt(i, 1);
000797          pObject := TPObject(SELF.AddrMember(i));
000798          pObject^ := object;
000799
000800          {$IFC fTrace}EP;{$ENDC}
000801      END;
000802
000803
000804  {$S SgABCdat}
000805      PROCEDURE TList.InsFirst(object: TObject);
000806      BEGIN
000807          {$IFC fTrace}BP(3);{$ENDC}
000808          SELF.InsAt(1, object);
000809          {$IFC fTrace}EP;{$ENDC}
000810      END;
000811
```

```
000812
000813  {$S sResDat}
000814      PROCEDURE TList.InsLast(object: TObject);
000815      BEGIN
000816          {$IFC fTrace}BP(3);{$ENDC}
000817          SELF.InsAt(SELF.size + 1, object);
000818          {$IFC fTrace}EP;{$ENDC}
000819      END;
000820
000821
000822  {$S sResDat}
000823      FUNCTION  TList.Last: TObject;
000824      BEGIN
000825          {$IFC fTrace}BP(3);{$ENDC}
000826          Last := SELF.At(SELF.size);
000827          {$IFC fTrace}EP;{$ENDC}
000828      END;
000829
000830
000831  {$S SgABCdat}
000832      FUNCTION  TList.ManyAt(i, howMany: LONGINT): TList;
000833          VAR list: TList;
000834      BEGIN
000835          {$IFC fTrace}BP(4);{$ENDC}
000836          list := TList.CREATE(NIL, SELF.Heap, howMany);
000837          list.InsManyAt(1, SELF, i, howMany);
000838          ManyAt := list;
000839          {$IFC fTrace}EP;{$ENDC}
000840      END;
000841
000842
000843  {$S sResDat}
000844      FUNCTION  TList.MemberBytes: INTEGER;
000845      BEGIN
000846          {$IFC fTrace}BP(3);{$ENDC}
000847          MemberBytes := 4;
000848          {$IFC fTrace}EP;{$ENDC}
000849      END;
000850
000851
000852  {$S SgABCdat}
000853      FUNCTION  TList.PopLast: TObject;
000854      BEGIN
000855          {$IFC fTrace}BP(4);{$ENDC}
000856          PopLast := SELF.Last;
000857          SELF.DelLast(FALSE);
000858          {$IFC fTrace}EP;{$ENDC}
000859      END;
```

```
000860
000861
000862  {$S sResDat}
000863      FUNCTION  TList.Pos(after: LONGINT; object: TObject): LONGINT;
000864          VAR y:  TObject;
000865              s:  TListScanner;
000866      BEGIN
000867          {$IFC fTrace}BP(3);{$ENDC}
000868          Pos := after;
000869          s := SELF.ScannerFrom(after, scanForward);
000870          WHILE s.Scan(y) DO
000871              IF object = y THEN
000872                  BEGIN
000873                  Pos := s.position;
000874                  s.Done;
000875                  END;
000876          {$IFC fTrace}EP;{$ENDC}
000877      END;
000878
000879
000880  {$S SgABCdat}
000881      PROCEDURE TList.PutAt(i: LONGINT; object: TObject; freeOld: BOOLEAN);
000882          VAR pObject:    TPObject;
000883              oldObject:  TObject;
000884      BEGIN
000885          {$IFC fTrace}BP(4);{$ENDC}
000886          {$IFC fCheckIndices}
000887          IF fCheckIndices THEN
000888              SELF.CheckIndex(i);
000889          {$ENDC}
000890
000891         {pObject := TPObject(SELF.AddrMember(i));   but for speed...}
000892
000893          IF i > SELF.holeStart THEN
000894              i := i + SELF.holeSize;
000895          pObject := TPObject(TpLONGINT(SELF)^ + SELF.dynStart + (4 * (i - 1)));
000896
000897          oldObject := pObject^;
000898          pObject^ := object;
000899
000900          IF freeOld THEN
000901              IF object <> oldObject THEN
000902                  Free(oldObject);
000903          {$IFC fTrace}EP;{$ENDC}
000904      END;
000905
000906
000907  {$S sResDat}
```

```
000908      FUNCTION  TList.Scanner: TListScanner;
000909      BEGIN
000910          {$IFC fTrace}BP(2);{$ENDC}
000911          Scanner := TListScanner.CREATE(NIL, SELF, 0, scanForward);
000912          {$IFC fTrace}EP;{$ENDC}
000913      END;
000914
000915
000916  {$S sResDat}
000917      FUNCTION  TList.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection): TListScanner;
000918      BEGIN
000919          {$IFC fTrace}BP(2);{$ENDC}
000920          ScannerFrom := TListScanner.CREATE(NIL, SELF, firstToScan, scanDirection);
000921          {$IFC fTrace}EP;{$ENDC}
000922      END;
000923
000924
000925  {$S sInit1}
000926  {$IFC compatibleLists} {For TIndexList.Class}
000927  BEGIN
000928      cList := THISCLASS;
000929  {$ENDC}
000930  END;
000931  {$S SgCLAres}
000932
000933
000934  METHODS OF TArray;
000935
000936
000937  {$S sResDat}
000938      FUNCTION  TArray.CREATE(object: TObject; heap: THeap; initialSlack, bytesPerRecord: INTEGER): TArray;
000939      BEGIN
000940          {$IFC fTrace}BP(1);{$ENDC}
000941          IF ODD(bytesPerRecord) THEN
000942              bytesPerRecord := bytesPerRecord + 1;
000943          IF object = NIL THEN
000944              object := NewDynObject(heap, THISCLASS, initialSlack * bytesPerRecord);
000945          SELF := TArray(TCollection.CREATE(object, heap, initialSlack));
000946          SELF.recordBytes := bytesPerRecord;
000947          {$IFC fTrace}EP;{$ENDC}
000948      END;
000949
000950
000951      {$IFC fDebugMethods}
000952      {$S SgCLAdbg}
000953      PROCEDURE TArray.Fields(PROCEDURE Field(nameAndType: S255));
000954      BEGIN
000955          SUPERSELF.Fields(Field);
```

```
000956          Field('recordBytes: INTEGER');
000957      END;
000958      {$S SgCLAres}
000959      {$ENDC}
000960
000961
000962      {$IFC fDebugMethods}
000963      {$S SgCLAdbg}
000964      PROCEDURE TArray.Debug(numLevels: INTEGER; memberTypeStr: S255);
000965          VAR s:          TArrayScanner;
000966              pRecord:    Ptr;
000967              i:          INTEGER;
000968              j:          INTEGER;
000969              str:        S255;
000970              hexOrd:     S8;
000971
000972          PROCEDURE SupplyMember(PROCEDURE Field(nameAndType: S255));
000973          BEGIN
000974              Field(Concat(str, ': ', memberTypeStr));
000975          END;
000976
000977      BEGIN
000978          SUPERSELF.Debug(numLevels, '');         { this prints other fields of the array }
000979          IF (numLevels > 1) OR ((numLevels = 1) AND (memberTypeStr <> '')) THEN
000980              BEGIN
000981              WrStr('{ ');
000982              i := 0;
000983              s := SELF.Scanner;
000984              IF s.position = SELF.holeStart THEN
000985                  WrStr(' <=HOLE=> ');
000986              WHILE s.Scan(pRecord) DO
000987                  BEGIN
000988                  IF i > 0 THEN
000989                      WrStr(', ');
000990                  i := i + 1;
000991                  IntToStr(i, @str);
000992                  IF memberTypeStr = '' THEN
000993                      BEGIN
000994                      str := CONCAT(str, ': ');
000995                      FOR j := 0 TO SELF.recordBytes-1 DO
000996                          BEGIN
000997                          LIntToHex(TPByte(ORD(pRecord)+j)^, @hexOrd);
000998                          str := CONCAT(str, Copy(hexOrd, 7, 2));
000999                          END;
001000                      WrStr(str);
001001                      END
001002                  ELSE
001003                      WriteDRecord(numLevels - 1, @pRecord, 0, SupplyMember);
```

```
001004                IF s.position = SELF.holeStart THEN
001005                    WrStr(', <=HOLE=> ');
001006                END;
001007            WrStr(' }');
001008            END;
001009    END;
001010    {$S SgCLAres}
001011    {$ENDC}
001012
001013
001014  {$S sResDat}
001015    FUNCTION  TArray.At(i: LONGINT): Ptr;
001016    BEGIN
001017        {$IFC fTrace}BP(3);{$ENDC}
001018        {$IFC fCheckIndices}
001019        IF fCheckIndices THEN
001020            SELF.CheckIndex(i);
001021        {$ENDC}
001022
001023       { At := Ptr(SELF.AddrMember(i));   but for speed...}
001024
001025        IF i > SELF.holeStart THEN
001026            i := i + SELF.holeSize;
001027
001028        At := Ptr(TpLONGINT(SELF)^ + SELF.dynStart + (SELF.recordBytes * (i - 1)));
001029        {$IFC fTrace}EP;{$ENDC}
001030    END;
001031
001032
001033  {$S SgABCdat}
001034    PROCEDURE TArray.DelAll;
001035    BEGIN
001036        {$IFC fTrace}BP(4);{$ENDC}
001037        SELF.EditAt(1, -SELF.size);
001038        {$IFC fTrace}EP;{$ENDC}
001039    END;
001040
001041
001042  {$S SgABCdat}
001043    PROCEDURE TArray.DelAt(i: LONGINT);
001044    BEGIN
001045        {$IFC fTrace}BP(4);{$ENDC}
001046        SELF.EditAt(i, -1);
001047        {$IFC fTrace}EP;{$ENDC}
001048    END;
001049
001050
001051  {$S SgABCdat}
```

```
001052      PROCEDURE TArray.DelFirst;
001053      BEGIN
001054          {$IFC fTrace}BP(3);{$ENDC}
001055          SELF.DelAt(1);
001056          {$IFC fTrace}EP;{$ENDC}
001057      END;
001058
001059
001060  {$S SgABCdat}
001061      PROCEDURE TArray.DelLast;
001062      BEGIN
001063          {$IFC fTrace}BP(3);{$ENDC}
001064          SELF.DelAt(SELF.size);
001065          {$IFC fTrace}EP;{$ENDC}
001066      END;
001067
001068
001069  {$S SgABCdat}
001070      PROCEDURE TArray.DelManyAt(i, howMany: LONGINT);
001071          VAR j: INTEGER;
001072      BEGIN
001073          {$IFC fTrace}BP(4);{$ENDC}
001074          IF howMany > 0 THEN
001075              SELF.EditAt(i, -howMany);
001076          {$IFC fTrace}EP;{$ENDC}
001077      END;
001078
001079
001080  {$S SgABCdat}
001081      PROCEDURE TArray.Each(PROCEDURE DoToRecord(pRecord: Ptr));
001082          VAR holeStart:      INTEGER;
001083              offset:         INTEGER;
001084              recordBytes:    INTEGER;
001085              j:              INTEGER;
001086      BEGIN
001087          {$IFC fTrace}BP(4);{$ENDC}
001088          holeStart := SELF.holeStart;
001089          offset := SELF.dynStart;
001090          recordBytes := SELF.recordBytes;
001091          FOR j := 0 TO SELF.size - 1 DO
001092              BEGIN
001093              IF j = holeStart THEN
001094                  offset := offset + recordBytes * SELF.holeSize;
001095              DoToRecord(Ptr(TpLONGINT(SELF)^ + offset));
001096              offset := offset + recordBytes;
001097              END;
001098          {$IFC fTrace}EP;{$ENDC}
001099      END;
```

```
001100
001101
001102  {$S SgABCdat}
001103      FUNCTION  TArray.First: Ptr;
001104      BEGIN
001105          {$IFC fTrace}BP(3);{$ENDC}
001106          First := SELF.At(1);
001107          {$IFC fTrace}EP;{$ENDC}
001108      END;
001109
001110
001111  {$S sResDat}
001112      PROCEDURE TArray.GetAt(i: LONGINT; pRecord: Ptr);
001113      BEGIN
001114          {$IFC fTrace}BP(4);{$ENDC}
001115          {$IFC fCheckIndices}
001116          IF fCheckIndices THEN
001117              SELF.CheckIndex(i);
001118          {$ENDC}
001119
001120          XferLeft(Ptr(SELF.AddrMember(i)), pRecord, SELF.recordBytes);
001121          {$IFC fTrace}EP;{$ENDC}
001122      END;
001123
001124
001125  {$S sResDat}
001126      PROCEDURE TArray.InsAt(i: LONGINT; pRecord: Ptr);
001127      BEGIN
001128          {$IFC fTrace}BP(4);{$ENDC}
001129          SELF.EditAt(i, 1);
001130          SELF.PutAt(i, pRecord);
001131          {$IFC fTrace}EP;{$ENDC}
001132      END;
001133
001134
001135  {$S sResDat}
001136      PROCEDURE TArray.InsFirst(pRecord: Ptr);
001137      BEGIN
001138          {$IFC fTrace}BP(3);{$ENDC}
001139          SELF.InsAt(1, pRecord);
001140          {$IFC fTrace}EP;{$ENDC}
001141      END;
001142
001143
001144  {$S sResDat}
001145      PROCEDURE TArray.InsLast(pRecord: Ptr);
001146      BEGIN
001147          {$IFC fTrace}BP(3);{$ENDC}
```

```
001148          SELF.InsAt(SELF.size + 1, pRecord);
001149          {$IFC fTrace}EP;{$ENDC}
001150      END;
001151
001152
001153  {$S SgABCdat}
001154      FUNCTION  TArray.Last: Ptr;
001155      BEGIN
001156          {$IFC fTrace}BP(3);{$ENDC}
001157          Last := SELF.At(SELF.size);
001158          {$IFC fTrace}EP;{$ENDC}
001159      END;
001160
001161
001162  {$S SgABCdat}
001163      FUNCTION  TArray.ManyAt(i, howMany: LONGINT): TArray;
001164          VAR arr: TArray;
001165      BEGIN
001166          {$IFC fTrace}BP(4);{$ENDC}
001167          arr := TArray.CREATE(NIL, SELF.Heap, howMany, SELF.recordBytes);
001168          arr.InsManyAt(1, SELF, i, howMany);
001169          ManyAt := arr;
001170          {$IFC fTrace}EP;{$ENDC}
001171      END;
001172
001173
001174  {$S sResDat}
001175      FUNCTION  TArray.MemberBytes: INTEGER;
001176      BEGIN
001177          {$IFC fTrace}BP(3);{$ENDC}
001178          MemberBytes := SELF.recordBytes;
001179          {$IFC fTrace}EP;{$ENDC}
001180      END;
001181
001182
001183  {$S SgABCdat}
001184      FUNCTION  TArray.Pos(after: LONGINT; pRecord: Ptr): LONGINT;
001185          VAR y:  Ptr;
001186              s:  TArrayScanner;
001187
001188          FUNCTION EqualRecords(p, q: Ptr; n: INTEGER): BOOLEAN; {n is even}
001189              VAR i:  INTEGER;
001190          BEGIN
001191              EqualRecords := FALSE;
001192              i := 0;
001193              WHILE i < n DO
001194                  BEGIN
001195                  IF TpINTEGER(ORD(p) + i)^ <> TpINTEGER(ORD(q) + i)^ THEN
```

```
001196                      EXIT(EqualRecords);
001197                  i := i + 2;
001198                  END;
001199              EqualRecords := TRUE;
001200          END;
001201
001202      BEGIN
001203          {$IFC fTrace}BP(3);{$ENDC}
001204          Pos := after;
001205          s := SELF.ScannerFrom(after, scanForward);
001206          WHILE s.Scan(y) DO
001207              IF EqualRecords(pRecord, y, SELF.recordBytes) THEN
001208                  BEGIN
001209                  Pos := s.position;
001210                  s.Done;
001211                  END;
001212          {$IFC fTrace}EP;{$ENDC}
001213      END;
001214
001215
001216  {$S sResDat}
001217      PROCEDURE TArray.PutAt(i: LONGINT; pRecord: Ptr);
001218      BEGIN
001219          {$IFC fTrace}BP(4);{$ENDC}
001220          {$IFC fCheckIndices}
001221          IF fCheckIndices THEN
001222              SELF.CheckIndex(i);
001223          {$ENDC}
001224
001225          XferLeft(pRecord, Ptr(SELF.AddrMember(i)), SELF.recordBytes);
001226          {$IFC fTrace}EP;{$ENDC}
001227      END;
001228
001229
001230  {$S SgABCdat}
001231      FUNCTION  TArray.Scanner: TArrayScanner;
001232      BEGIN
001233          {$IFC fTrace}BP(2);{$ENDC}
001234          Scanner := TArrayScanner.CREATE(NIL, SELF, 0, scanForward);
001235          {$IFC fTrace}EP;{$ENDC}
001236      END;
001237
001238
001239  {$S SgABCdat}
001240      FUNCTION  TArray.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection): TArrayScanner;
001241      BEGIN
001242          {$IFC fTrace}BP(2);{$ENDC}
001243          ScannerFrom := TArrayScanner.CREATE(NIL, SELF, firstToScan, scanDirection);
```

```
001244          {$IFC fTrace}EP;{$ENDC}
001245      END;
001246
001247
001248  {$S sInit1}
001249  {$IFC compatibleLists} {For TDynamicArray.Class}
001250  BEGIN
001251      cArray := THISCLASS;
001252  {$ENDC}
001253  END;
001254
001255
001256  METHODS OF TString;
001257
001258
001259  {$S sResDat}
001260      FUNCTION  TString.CREATE(object: TObject; heap: THeap; initialSlack: INTEGER): TString;
001261      BEGIN
001262          {$IFC fTrace}BP(1);{$ENDC}
001263          IF ODD(initialSlack) THEN
001264              initialSlack := initialSlack + 1;
001265          IF object = NIL THEN
001266              object := NewDynObject(heap, THISCLASS, initialSlack);
001267          SELF := TString(TCollection.CREATE(object, heap, initialSlack));
001268          {$IFC fTrace}EP;{$ENDC}
001269      END;
001270
001271
001272      {$IFC fDebugMethods}
001273      {$S SgCLAdbg}
001274      PROCEDURE TString.Debug(numLevels: INTEGER; memberTypeStr: S255);
001275      VAR s:          TStringScanner;
001276          ch:         CHAR;
001277          str:        S8;
001278      BEGIN
001279          SUPERSELF.Debug(numLevels, '');         { this prints other fields of the list }
001280          IF numLevels > 0 THEN
001281              BEGIN
001282              WrStr('''');
001283              s := SELF.Scanner;
001284              IF s.position = SELF.holeStart THEN
001285                  WrStr('<=HOLE=>');
001286              str := 'x';
001287              WHILE s.Scan(ch) DO
001288                  BEGIN
001289                  str[1] := ch;
001290                  WrStr(str);
001291                  IF s.position = SELF.holeStart THEN
```

```
001292                  WrStr('<=HOLE=>');
001293              END;
001294          WrStr('''');
001295          END;
001296      END;
001297      {$S SgCLAres}
001298      {$ENDC}
001299
001300
001301  {$S SgCLAres}
001302      FUNCTION  TString.At(i: LONGINT): CHAR;
001303      BEGIN
001304          {$IFC fTrace}BP(3);{$ENDC}
001305          {$IFC fCheckIndices}
001306          IF fCheckIndices THEN
001307              SELF.CheckIndex(i);
001308          {$ENDC}
001309
001310          IF i > SELF.holeStart THEN
001311              i := i + SELF.holeSize;
001312          At := TpPAOC(TpLONGINT(SELF)^ + SELF.dynStart)^[i];
001313          {$IFC fTrace}EP;{$ENDC}
001314      END;
001315
001316
001317  {$S SgCLAres}
001318      PROCEDURE TString.DelAll;
001319      BEGIN
001320          {$IFC fTrace}BP(4);{$ENDC}
001321          SELF.EditAt(1, -SELF.size);
001322          {$IFC fTrace}EP;{$ENDC}
001323      END;
001324
001325
001326  {$S SgCLAres}
001327      PROCEDURE TString.DelAt(i: LONGINT);
001328      BEGIN
001329          {$IFC fTrace}BP(4);{$ENDC}
001330          SELF.EditAt(i, -1);
001331          {$IFC fTrace}EP;{$ENDC}
001332      END;
001333
001334
001335  {$S SgCLAres}
001336      PROCEDURE TString.DelFirst;
001337      BEGIN
001338          {$IFC fTrace}BP(3);{$ENDC}
001339          SELF.DelAt(1);
```

```
001340          {$IFC fTrace}EP;{$ENDC}
001341      END;
001342
001343
001344  {$S SgCLAres}
001345      PROCEDURE TString.DelLast;
001346      BEGIN
001347          {$IFC fTrace}BP(3);{$ENDC}
001348          SELF.DelAt(SELF.size);
001349          {$IFC fTrace}EP;{$ENDC}
001350      END;
001351
001352
001353  {$S SgCLAres}
001354      PROCEDURE TString.DelManyAt(i, howMany: LONGINT);
001355          VAR j: INTEGER;
001356      BEGIN
001357          {$IFC fTrace}BP(4);{$ENDC}
001358          IF howMany > 0 THEN
001359              SELF.EditAt(i, -howMany);
001360          {$IFC fTrace}EP;{$ENDC}
001361      END;
001362
001363
001364  {$S SgCLAres}
001365      PROCEDURE TString.Draw(i: LONGINT; howMany: INTEGER);
001366          VAR beforeHole: INTEGER;
001367              pWord1:     TpINTEGER;
001368      BEGIN
001369          {$IFC fTrace}BP(4);{$ENDC}
001370          beforeHole := Min(SELF.holeStart - (i - 1), howMany);
001371          pWord1 := TpINTEGER(TpLONGINT(SELF)^ + SELF.dynStart);
001372          IF beforeHole > 0 THEN
001373              DrawLText(pWord1, i - 1, beforeHole);
001374          IF beforeHole < howMany THEN
001375              DrawLText(pWord1, SELF.holeStart + SELF.holeSize - Min(beforeHole, 0),
001376                                howMany - Max(beforeHole, 0));
001377          {$IFC fTrace}EP;{$ENDC}
001378      END;
001379
001380
001381  {$S sResDat}
001382      FUNCTION  TString.Width(i: LONGINT; howMany: INTEGER): INTEGER;
001383          VAR beforeHole: INTEGER;
001384              pWord1:     TpINTEGER;
001385              totalWidth: INTEGER;
001386      BEGIN
001387          {$IFC fTrace}BP(4);{$ENDC}
```

```
001388          beforeHole := Min(SELF.holeStart - (i - 1), howMany);
001389          pWord1 := TpINTEGER(TpLONGINT(SELF)^ + SELF.dynStart);
001390          totalWidth := 0;
001391          IF beforeHole > 0 THEN
001392              totalWidth := TextWidth(pWord1, i - 1, beforeHole);
001393          IF beforeHole < howMany THEN
001394              totalWidth := totalWidth + TextWidth(pWord1, SELF.holeStart + SELF.holeSize - Min(beforeHole, 0),
001395                                                   howMany - Max(beforeHole, 0));
001396          Width := totalWidth;
001397          {$IFC fTrace}EP;{$ENDC}
001398      END;
001399
001400
001401  {$S SgCLAres}
001402      PROCEDURE TString.Each(PROCEDURE DoToCharacter(character: CHAR));
001403          VAR holeStart:      INTEGER;
001404              offset:         INTEGER;
001405              j:              INTEGER;
001406              pChars:         TpPAOC;
001407      BEGIN
001408          {$IFC fTrace}BP(4);{$ENDC}
001409          holeStart := SELF.holeStart;
001410          pChars := TpPAOC(TpLONGINT(SELF)^ + SELF.dynStart);
001411          offset := 1;
001412          FOR j := 0 TO SELF.size - 1 DO
001413              BEGIN
001414              IF j = holeStart THEN
001415                  offset := offset + SELF.holeSize;
001416              DoToCharacter(pChars^[offset]);
001417              offset := offset + 1;
001418              END;
001419          {$IFC fTrace}EP;{$ENDC}
001420      END;
001421
001422
001423  {$S SgCLAres}
001424      FUNCTION  TString.First: CHAR;
001425      BEGIN
001426          {$IFC fTrace}BP(3);{$ENDC}
001427          First := SELF.At(1);
001428          {$IFC fTrace}EP;{$ENDC}
001429      END;
001430
001431
001432  {$S SgCLAres}
001433      PROCEDURE TString.InsAt(i: LONGINT; character: CHAR);
001434          VAR pPAOC:      TpPAOC;
001435      BEGIN
```

```
001436          {$IFC fTrace}BP(4);{$ENDC}
001437          SELF.EditAt(i, 1);
001438
001439          pPAOC := TpPAOC(TpLONGINT(SELF)^ + SELF.dynStart);
001440          pPAOC^[i] := character;
001441          {$IFC fTrace}EP;{$ENDC}
001442      END;
001443
001444
001445  {$S SgCLAres}
001446      PROCEDURE TString.InsFirst(character: CHAR);
001447      BEGIN
001448          {$IFC fTrace}BP(3);{$ENDC}
001449          SELF.InsAt(1, character);
001450          {$IFC fTrace}EP;{$ENDC}
001451      END;
001452
001453
001454  {$S SgCLAres}
001455      PROCEDURE TString.InsLast(character: CHAR);
001456      BEGIN
001457          {$IFC fTrace}BP(3);{$ENDC}
001458          SELF.InsAt(SELF.size + 1, character);
001459          {$IFC fTrace}EP;{$ENDC}
001460      END;
001461
001462
001463  {$S sResDat}
001464      PROCEDURE TString.InsPStrAt(i: LONGINT; pStr: TPString);
001465      BEGIN
001466          {$IFC fTrace}BP(3);{$ENDC}
001467          SELF.EditAt(i, Length(pStr^));
001468          XferLeft(Ptr(ORD(pStr)+1), Ptr(SELF.AddrMember(i)), Length(pStr^));
001469          {$IFC fTrace}EP;{$ENDC}
001470      END;
001471
001472
001473  {$S SgCLAres}
001474      FUNCTION  TString.Last: CHAR;
001475      BEGIN
001476          {$IFC fTrace}BP(3);{$ENDC}
001477          Last := SELF.At(SELF.size);
001478          {$IFC fTrace}EP;{$ENDC}
001479      END;
001480
001481
001482  {$S SgCLAres}
001483      FUNCTION  TString.ManyAt(i, howMany: LONGINT): TString;
```

```
001484          VAR str: TString;
001485      BEGIN
001486          {$IFC fTrace}BP(4);{$ENDC}
001487          str := TString.CREATE(NIL, SELF.Heap, howMany);
001488          str.InsManyAt(1, SELF, i, howMany);
001489          ManyAt := str;
001490          {$IFC fTrace}EP;{$ENDC}
001491      END;
001492
001493
001494  {$S sResDat}
001495      FUNCTION  TString.MemberBytes: INTEGER;
001496      BEGIN
001497          {$IFC fTrace}BP(3);{$ENDC}
001498          MemberBytes := 1;
001499          {$IFC fTrace}EP;{$ENDC}
001500      END;
001501
001502
001503  {$S SgCLAres}
001504      FUNCTION  TString.Pos(after: LONGINT; character: CHAR): LONGINT;
001505          VAR y:  CHAR;
001506              s:  TStringScanner;
001507      BEGIN
001508          {$IFC fTrace}BP(3);{$ENDC}
001509          Pos := after;
001510          s := SELF.ScannerFrom(after, scanForward);
001511          WHILE s.Scan(y) DO
001512              IF y = character THEN
001513                  BEGIN
001514                  Pos := s.position;
001515                  s.Done;
001516                  END;
001517          {$IFC fTrace}EP;{$ENDC}
001518      END;
001519
001520
001521  {$S SgCLAres}
001522      PROCEDURE TString.PutAt(i: LONGINT; character: CHAR);
001523          VAR pPAOC:  TpPAOC;
001524      BEGIN
001525          {$IFC fTrace}BP(4);{$ENDC}
001526          {$IFC fCheckIndices}
001527          IF fCheckIndices THEN
001528              SELF.CheckIndex(i);
001529          {$ENDC}
001530
001531          IF i > SELF.holeStart THEN
```

```
001532              i := i + SELF.holeSize;
001533
001534          pPAOC := TpPAOC(TpLONGINT(SELF)^ + SELF.dynStart);
001535          pPAOC^[i] := character;
001536          {$IFC fTrace}EP;{$ENDC}
001537      END;
001538
001539
001540  {$S SgCLAres}
001541      FUNCTION  TString.Scanner: TStringScanner;
001542      BEGIN
001543          {$IFC fTrace}BP(2);{$ENDC}
001544          Scanner := TStringScanner.CREATE(NIL, SELF, 0, scanForward);
001545          {$IFC fTrace}EP;{$ENDC}
001546      END;
001547
001548
001549  {$S SgCLAres}
001550      FUNCTION  TString.ScannerFrom(firstToScan: LONGINT; scanDirection: TScanDirection): TStringScanner;
001551      BEGIN
001552          {$IFC fTrace}BP(2);{$ENDC}
001553          ScannerFrom := TStringScanner.CREATE(NIL, SELF, firstToScan, scanDirection);
001554          {$IFC fTrace}EP;{$ENDC}
001555      END;
001556
001557
001558  {$S SgCLAres}
001559      PROCEDURE TString.ToPStr(pStr: TPString);
001560      BEGIN
001561          {$IFC fTrace}BP(3);{$ENDC}
001562          SELF.ToPStrAt(1, SELF.size, pStr);
001563          {$IFC fTrace}EP;{$ENDC}
001564      END;
001565
001566
001567  {$S SgCLAres}
001568      PROCEDURE TString.ToPStrAt(i, howMany: LONGINT; pStr: TPString);
001569      BEGIN
001570          {$IFC fTrace}BP(3);{$ENDC}
001571          {$IFC fCheckIndices}
001572          IF howMany > 255 THEN
001573              ABCBreak('ToPStrAt: Too many characters', howMany);
001574          {$ENDC}
001575          SELF.EditAt(i + howMany, 0);
001576          XferLeft(Ptr(SELF.AddrMember(i)), Ptr(ORD(pStr)+1), howMany);
001577    {$R-} pStr^[0] := CHAR(howMany); {$IFC fRngObject}{$R+}{$ENDC}
001578          {$IFC fTrace}EP;{$ENDC}
001579      END;
```

```
001580
001581
001582  {$S sInit1}
001583  END;
001584  {$S SgCLAres}
001585
001586
001587  METHODS OF TFile;
001588
001589
001590  {$S sResDat}
001591      FUNCTION  TFile.CREATE(object: TObject; heap: THeap; itsPath: TFilePath;
001592                             itsPassword: TPassword): TFile;
001593          VAR pPath:          TPPathname;
001594              error:          INTEGER;
001595          {$IFC LibraryVersion <= 20}
001596              fsInfo:         FS_Info;
001597          {$ELSEC}
001598              fsInfo:         Q_Info;
001599          {$ENDC}
001600              itsScanners:    TList;
001601      BEGIN
001602          {$IFC fTrace}BP(1);{$ENDC}
001603          IF object = NIL THEN
001604              object := NewObject(heap, THISCLASS);
001605          SELF := TFile(TCollection.CREATE(object, heap, 0)); {Just to initialize those ignored fields}
001606          pPath := @itsPath;
001607        {$IFC LibraryVersion <= 20}
001608          Lookup(error, pPath^, fsInfo);
001609        {$ELSEC}
001610          Quick_Lookup(error, pPath^, fsInfo);
001611        {$ENDC}
001612
001613          itsScanners := TList.CREATE(NIL, heap, 0);
001614          WITH SELF DO
001615              BEGIN
001616              dynStart := MAXINT;
001617              IF error > 0 THEN
001618                  size := 0
001619              ELSE
001620                  size := fsInfo.size;
001621              path := itsPath;
001622              password := itsPassword;
001623              scanners := itsScanners;
001624              END;
001625          {$IFC fTrace}EP;{$ENDC}
001626      END;
001627
```

```
001628
001629  {$S sResDat}
001630      PROCEDURE TFile.Free;                                {Free frees the scanners as well}
001631      BEGIN
001632          {$IFC fTrace}BP(5);{$ENDC}
001633          SELF.scanners.Free;
001634          SUPERSELF.Free;
001635          {$IFC fTrace}EP;{$ENDC}
001636      END;
001637      {$S SgCLAres}
001638
001639
001640      {$IFC fDbgObject}
001641      {$S SgCLAdbg}
001642      FUNCTION  TFile.Clone(heap: THeap): TObject;
001643      BEGIN
001644          ABCBreak('A TFile cannot Clone', 0);
001645      END;
001646      {$S SgCLAres}
001647      {$ENDC}
001648
001649
001650      {$IFC fDebugMethods}
001651      {$S SgCLAdbg}
001652      PROCEDURE TFile.Fields(PROCEDURE Field(nameAndType: S255));
001653      BEGIN
001654          SUPERSELF.Fields(Field);
001655          Field('path: STRING[255]');
001656          Field('password: STRING[32]');
001657          Field('scanners: TList');
001658      END;
001659      {$S SgCLAres}
001660      {$ENDC}
001661
001662
001663      {$S SgCLAcld}
001664      PROCEDURE TFile.ChangePassword(VAR error: INTEGER; newPassword: TPassword);
001665          VAR pPath:      TPPathname;
001666              pPass:      TPEName;
001667              pNPass:     TPEName;
001668      BEGIN
001669          {$IFC fMaxTrace}BP(1);{$ENDC}
001670          {$IFC fMaxTrace}EP;{$ENDC}
001671          {$IFC LibraryVersion <= 20}
001672          error := -1293; {warning: file is not password protected}
001673          {$ELSEC}
001674          pPath := @SELF.path;
001675          pPass := @SELF.password;
```

```
001676          pNPass := @newPassword;
001677          Change_Password(error, pPath^, pPass^, pNPass^);
001678          {$ENDC}
001679          IF error <= 0 THEN
001680              SELF.password := newPassword;
001681      END;
001682      {$S SgCLAres}
001683
001684
001685      {$S SgCLAcld}
001686      PROCEDURE TFile.Delete(VAR error: INTEGER);
001687          VAR pPath:      TPPathname;
001688              {$IFC LibraryVersion > 20}
001689              pPass:      TPEName;
001690              {$ENDC}
001691      BEGIN
001692          {$IFC fMaxTrace}BP(1);{$ENDC}
001693          {$IFC fMaxTrace}EP;{$ENDC}
001694          pPath := @SELF.path;
001695          {$IFC LibraryVersion <= 20}
001696          Kill_Object(error, pPath^);
001697          {$ELSEC}
001698          pPass := @SELF.password;
001699          Kill_Secure(error, pPath^, pPass^);
001700          {$ENDC}
001701      END;
001702      {$S SgCLAres}
001703
001704
001705      {$S sResDat}
001706      FUNCTION TFile.Exists(VAR error: INTEGER): BOOLEAN;
001707        {$IFC LibraryVersion <= 20}
001708          VAR refInfo:    FS_Info;
001709        {$ELSEC}
001710          VAR refInfo:    Q_Info;
001711        {$ENDC}
001712              pPath:      TPPathname;
001713      BEGIN
001714          {$IFC fMaxTrace}BP(1);{$ENDC}
001715          {$IFC fMaxTrace}EP;{$ENDC}
001716          pPath := @SELF.path;
001717        {$IFC LibraryVersion <= 20}
001718          Lookup(error, pPath^, refInfo);
001719        {$ELSEC}
001720          Quick_Lookup(error, pPath^, refInfo);
001721        {$ENDC}
001722          Exists := error <= 0;
001723      END;
```

```
001724      {$S SgCLAres}
001725
001726
001727  {$S SgABCdat}
001728      FUNCTION  TFile.MemberBytes: INTEGER;
001729      BEGIN
001730          {$IFC fTrace}BP(3);{$ENDC}
001731          MemberBytes := 1;
001732          {$IFC fTrace}EP;{$ENDC}
001733      END;
001734
001735
001736      {$S SgCLAcld}
001737      PROCEDURE TFile.Rename(VAR error: INTEGER; newFileName: TFilePath);
001738              {the volume of newFileName is ignored}
001739          VAR pPath:          TPPathname;
001740              vol:            TFilePath;
001741              name:           TFilePath;
001742              pEName:         TPEname;
001743              {$IFC LibraryVersion > 20}
001744              pPass:          TPEName;
001745              {$ENDC}
001746      BEGIN
001747          {$IFC fMaxTrace}BP(1);{$ENDC}
001748          {$IFC fMaxTrace}EP;{$ENDC}
001749          pPath := @SELF.path;
001750          SplitFilePath(newFileName, vol, name);
001751          pEName := @name;
001752          {$IFC LibraryVersion <= 20}
001753          Rename_Entry(error, pPath^, pEName^);
001754          {$ELSEC}
001755          pPass := @SELF.password;
001756          Rename_Secure(error, pPath^, pEName^, pPass^);
001757          {$ENDC}
001758      END;
001759      {$S SgCLAres}
001760
001761
001762      {$S SgCLAcld}
001763      FUNCTION  TFile.Scanner: TFileScanner;
001764      BEGIN
001765          {$IFC fTrace}BP(2);{$ENDC}
001766          Scanner := SELF.ScannerFrom(0, [fRead, fWrite]);
001767          {$IFC fTrace}EP;{$ENDC}
001768      END;
001769      {$S SgCLAres}
001770
001771
```

```
001772  {$S sResDat}
001773      FUNCTION  TFile.ScannerFrom(firstToScan: LONGINT; manip: TAccesses): TFileScanner;
001774          VAR s: TFileScanner;
001775      BEGIN
001776          {$IFC fTrace}BP(2);{$ENDC}
001777          s := TFileScanner.CREATE(NIL, SELF, manip);
001778          s.Seek(firstToScan);
001779          ScannerFrom := s;
001780          {$IFC fTrace}EP;{$ENDC}
001781      END;
001782      {$S SgCLAres}
001783
001784
001785      {$S SgCLAcld}
001786      FUNCTION  TFile.VerifyPassword(VAR error: INTEGER; password: TPassword): BOOLEAN;
001787          VAR pPath:  TPPathname;
001788              pPass:  TPEName;
001789      BEGIN
001790          {$IFC fMaxTrace}BP(1);{$ENDC}
001791          {$IFC fMaxTrace}EP;{$ENDC}
001792          {$IFC LibraryVersion <= 20}
001793          error := -1293; {warning file is not password protected}
001794          VerifyPassword := TRUE;
001795          {$ELSEC}
001796          pPath := @SELF.path;
001797          pPass := @password;
001798          Verify_Password(error, pPath^, pPass^);
001799          VerifyPassword := error <= 0;
001800          {$ENDC}
001801      END;
001802      {$S SgCLAres}
001803
001804
001805      {$S SgCLAcld}
001806      FUNCTION TFile.WhenModified(VAR error: INTEGER): LONGINT;
001807        {$IFC LibraryVersion <= 20}
001808          VAR refInfo:    FS_Info;
001809        {$ELSEC}
001810          VAR refInfo:    Q_Info;
001811        {$ENDC}
001812              pPath:      TPPathname;
001813      BEGIN
001814          {$IFC fMaxTrace}BP(1);{$ENDC}
001815          {$IFC fMaxTrace}EP;{$ENDC}
001816          pPath := @SELF.path;
001817        {$IFC LibraryVersion <= 20}
001818          Lookup(error, pPath^, refInfo);
001819        {$ELSEC}
```

```
001820          Quick_Lookup(error, pPath^, refInfo);
001821        {$ENDC}
001822          IF error <= 0 THEN
001823              WhenModified := refInfo.DTM
001824          ELSE
001825              WhenModified := -1;
001826      END;
001827      {$S SgCLAres}
001828
001829
001830  {$S sInit1}
001831  END;
001832  {$S SgCLAres}
001833
001834
001835  METHODS OF TScanner;
001836
001837
001838  {$S sResDat}
001839      FUNCTION  TScanner.CREATE(object: TObject; itsCollection: TCollection;
001840                                itsInitialPosition: LONGINT; scanDirection: TScanDirection): TScanner;
001841      BEGIN
001842          {$IFC fTrace}BP(1);{$ENDC}
001843          IF object = NIL THEN
001844              ABCBreak('TScanner.CREATE must be passed an already-allocated object by a subclass CREATE', 0);
001845          SELF := TScanner(object);
001846          WITH SELF DO
001847              BEGIN
001848              collection := itsCollection;
001849        {$H-} position := Max(0, Min(collection.size+1, itsInitialPosition)); {$H+}
001850              scanDone := FALSE;
001851
001852              IF scanDirection = scanForward THEN
001853                  BEGIN
001854                  increment := 1;
001855                  atEnd := position >= collection.size;
001856                  END
001857              ELSE
001858                  BEGIN
001859                  increment := -1;
001860                  atEnd := position <= 1;
001861                  END;
001862              END;
001863          SELF.Seek(itsInitialPosition);
001864          {$IFC fTrace}EP;{$ENDC}
001865      END;
001866
001867
```

```
001868      {$IFC fDebugMethods}
001869      {$S SgCLAdbg}
001870      PROCEDURE TScanner.Fields(PROCEDURE Field(nameAndType: S255));
001871      BEGIN
001872          SUPERSELF.Fields(Field);
001873          Field('collection: TCollection');
001874          Field('position: LONGINT');
001875          Field('increment: INTEGER');
001876          Field('scanDone: BOOLEAN');
001877          Field('atEnd: BOOLEAN');
001878      END;
001879      {$S SgCLAres}
001880      {$ENDC}
001881
001882
001883  {$S SgABCdat}
001884      FUNCTION  TScanner.Advance(PROCEDURE DoToCurrent(anotherMember: BOOLEAN)): BOOLEAN;
001885          VAR moreToScan: BOOLEAN;
001886      BEGIN
001887          {$IFC fTrace}BP(1);{$ENDC}
001888          WITH SELF DO
001889              IF scanDone THEN
001890                  moreToScan := FALSE     {don't reassign nextObject}
001891              ELSE
001892                  BEGIN
001893                  IF atEnd THEN
001894                      moreToScan := FALSE
001895                  ELSE
001896                      BEGIN
001897                      moreToScan := TRUE;
001898                      position := position + increment;
001899                      IF increment > 0 THEN
001900                          atEnd := position >= collection.size
001901                      ELSE
001902                          atEnd := position <= 1;
001903                      END;
001904
001905            {$H-} DoToCurrent(moreToScan); {$H+}
001906                  END;
001907
001908          IF NOT moreToScan THEN
001909              SELF.Free;
001910
001911          Advance := moreToScan;
001912          {$IFC fTrace}EP;{$ENDC}
001913      END;
001914
001915
```

```
001916  {$S SgABCdat}
001917      PROCEDURE TScanner.Allocate(slack: LONGINT);
001918      BEGIN
001919          {$IFC fTrace}BP(2);{$ENDC}
001920          SELF.collection.StartEdit(slack);
001921          {$IFC fTrace}EP;{$ENDC}
001922      END;
001923
001924
001925  {$S SgABCdat}
001926      PROCEDURE TScanner.Close;
001927      BEGIN
001928          {$IFC fTrace}BP(2);{$ENDC}
001929          {$IFC fTrace}EP;{$ENDC}
001930      END;
001931
001932
001933  {$S SgABCdat}
001934      PROCEDURE TScanner.Compact;
001935      BEGIN
001936          {$IFC fTrace}BP(2);{$ENDC}
001937          SELF.collection.StopEdit;
001938          {$IFC fTrace}EP;{$ENDC}
001939      END;
001940
001941
001942  {$S sResDat}
001943      PROCEDURE TScanner.Done;
001944      BEGIN
001945          {$IFC fTrace}BP(2);{$ENDC}
001946          SELF.scanDone := TRUE;
001947          {$IFC fTrace}EP;{$ENDC}
001948      END;
001949
001950
001951  {$S SgABCdat}
001952      PROCEDURE TScanner.Open;
001953      BEGIN
001954          {$IFC fTrace}BP(2);{$ENDC}
001955          {$IFC fTrace}EP;{$ENDC}
001956      END;
001957
001958
001959  {$S SgABCdat}
001960      PROCEDURE TScanner.Reverse;
001961      BEGIN
001962          {$IFC fTrace}BP(2);{$ENDC}
001963          SELF.increment := - SELF.increment;
```

```
001964          {$IFC fTrace}EP;{$ENDC}
001965      END;
001966
001967
001968  {$S sResDat}
001969      PROCEDURE TScanner.Seek(newPosition: LONGINT);
001970      BEGIN
001971          {$IFC fTrace}BP(2);{$ENDC}
001972          WITH SELF DO
001973              BEGIN
001974        {$H-} position := Max(0, Min(collection.size+1, newPosition)); {$H+}
001975              atEnd := ((position >= collection.size) AND (increment > 0)) OR
001976                       ((position <= 1) AND (increment < 0));
001977              END;
001978          {$IFC fTrace}EP;{$ENDC}
001979      END;
001980
001981
001982  {$S SgABCdat}
001983      PROCEDURE TScanner.Skip(deltaPos: LONGINT);
001984      BEGIN
001985          {$IFC fTrace}BP(2);{$ENDC}
001986          SELF.Seek(SELF.position + deltaPos);
001987          {$IFC fTrace}EP;{$ENDC}
001988      END;
001989
001990
001991  {$S sInit1}
001992  END;
001993  {$S SgCLAres}
001994
001995
001996  METHODS OF TListScanner;
001997
001998
001999  {$S sResDat}
002000      FUNCTION  TListScanner.CREATE(object: TObject; itsList: TList;
002001                                    itsInitialPosition: LONGINT; itsScanDirection: TScanDirection)
002002                                    : TListScanner;
002003      BEGIN
002004          {$IFC fTrace}BP(1);{$ENDC}
002005          IF object = NIL THEN
002006              object := NewOrRecycledObject(mainHeap, THISCLASS, availListScanner);
002007          SELF := TListScanner(TScanner.CREATE(object, itsList, itsInitialPosition, itsScanDirection));
002008          {$IFC fTrace}EP;{$ENDC}
002009      END;
002010
002011
```

```
002012  {$S sResDat}
002013      PROCEDURE TListScanner.Free;
002014      BEGIN
002015          {$IFC fTrace}BP(1);{$ENDC}
002016          RecycleObject(SELF, availListScanner);
002017          {$IFC fTrace}EP;{$ENDC}
002018      END;
002019
002020
002021  {$S SgABCdat}
002022      PROCEDURE TListScanner.Append(object: TObject);
002023      BEGIN
002024          {$IFC fTrace}BP(2);{$ENDC}
002025          TList(SELF.collection).InsAt(SELF.position + 1, object);
002026          SELF.position := SELF.position + 1;
002027
002028          (***** removed the following line: .InsAt should have set the collection size
002029    {$H-} SELF.collection.size := Max(SELF.collection.size, SELF.position); {$H+}
002030          *****)
002031          {$IFC fTrace}EP;{$ENDC}
002032      END;
002033
002034
002035  {$S SgABCdat}
002036      PROCEDURE TListScanner.Delete(freeOld: BOOLEAN);
002037      BEGIN
002038          {$IFC fTrace}BP(2);{$ENDC}
002039          TList(SELF.collection).DelAt(SELF.position, freeOld);
002040          WITH SELF DO
002041              IF increment > 0 THEN
002042                  position := position - 1;
002043          {$IFC fTrace}EP;{$ENDC}
002044      END;
002045
002046
002047  {$S SgABCdat}
002048      PROCEDURE TListScanner.DeleteRest(freeOld: BOOLEAN);
002049      BEGIN
002050          {$IFC fTrace}BP(2);{$ENDC}
002051          WITH SELF DO
002052              IF increment > 0 THEN
002053         {$H-}    TList(collection).DelManyAt(position + 1, collection.size - position, freeOld)
002054              ELSE
002055                  TList(collection).DelManyAt(1, position - 1, freeOld);  {$H+}
002056          WITH SELF DO
002057              BEGIN
002058              collection.size := position;
002059              atEnd := TRUE;
```

```
002060              END;
002061          {$IFC fTrace}EP;{$ENDC}
002062      END;
002063
002064
002065  {$S SgABCdat}
002066      FUNCTION  TListScanner.Obtain: TObject;
002067      BEGIN
002068          {$IFC fTrace}BP(1);{$ENDC}
002069          Obtain := TList(SELF.collection).At(SELF.position);
002070          {$IFC fTrace}EP;{$ENDC}
002071      END;
002072
002073
002074  {$S SgABCdat}
002075      PROCEDURE TListScanner.Replace(object: TObject; freeOld: BOOLEAN);
002076      BEGIN
002077          {$IFC fTrace}BP(2);{$ENDC}
002078          TList(SELF.collection).PutAt(SELF.position, object, freeOld);
002079          {$IFC fTrace}EP;{$ENDC}
002080      END;
002081
002082
002083  {$S sResDat}
002084      FUNCTION  TListScanner.Scan(VAR nextObject: TObject): BOOLEAN;
002085          VAR actIndex:   LONGINT;   {an actual index into the list, INCLUDING the hole as part of the list}
002086  (*
002087          PROCEDURE AssignListScanVariable(anotherObject: BOOLEAN);
002088          BEGIN
002089              IF anotherObject THEN
002090                  nextObject := TList(SELF.collection).At(SELF.position)
002091              ELSE
002092                  nextObject := NIL;
002093          END;
002094
002095      BEGIN
002096          {$IFC fTrace}BP(1);{$ENDC}
002097          Scan := SELF.Advance(AssignListScanVariable);
002098          {$IFC fTrace}EP;{$ENDC}
002099      END;
002100  *)
002101          VAR moreToScan: BOOLEAN;
002102      BEGIN {speedier version}
002103          {$IFC fTrace}BP(1);{$ENDC}
002104          WITH SELF DO
002105              IF scanDone THEN
002106                  moreToScan := FALSE     {don't reassign nextObject}
002107              ELSE
```

```
002108                  BEGIN
002109                  IF atEnd THEN
002110                      moreToScan := FALSE
002111                  ELSE
002112                      BEGIN
002113                      moreToScan := TRUE;
002114                      position := position + increment;
002115                      IF increment > 0 THEN
002116                          atEnd := position >= collection.size
002117                      ELSE
002118                          atEnd := position <= 1;
002119                      END;
002120
002121                  IF moreToScan THEN
002122                      BEGIN
002123                      IF position > collection.holeStart THEN
002124                          actIndex := position + collection.holeSize
002125                      ELSE
002126                          actIndex := position;
002127                      nextObject := TPObject(TpLONGINT(collection)^ + collection.dynStart
002128                                      + (4 * (actIndex - 1)))^;
002129                      END
002130                  ELSE
002131                      nextObject := NIL;
002132                  END;
002133
002134          IF NOT moreToScan THEN
002135              SELF.Free;
002136
002137          Scan := moreToScan;
002138          {$IFC fTrace}EP;{$ENDC}
002139      END;
002140
002141
002142  {$S sInit1}
002143  BEGIN
002144
002145      availListScanner := NIL;
002146
002147  END;
002148  {$S SgCLAres}
002149
002150
002151  METHODS OF TArrayScanner;
002152
002153
002154  {$S SgABCdat}
002155      FUNCTION  TArrayScanner.CREATE(object: TObject; itsArray: TArray;
```

```
002156                                  itsInitialPosition: LONGINT; itsScanDirection: TScanDirection)
002157                                  : TArrayScanner;
002158      BEGIN
002159          {$IFC fTrace}BP(1);{$ENDC}
002160          IF object = NIL THEN
002161              object := NewOrRecycledObject(mainHeap, THISCLASS, availArrayScanner);
002162          SELF := TArrayScanner(TScanner.CREATE(object, itsArray, itsInitialPosition, itsScanDirection));
002163          {$IFC fTrace}EP;{$ENDC}
002164      END;
002165
002166
002167  {$S SgABCdat}
002168      PROCEDURE TArrayScanner.Free;
002169      BEGIN
002170          {$IFC fTrace}BP(1);{$ENDC}
002171          RecycleObject(SELF, availArrayScanner);
002172          {$IFC fTrace}EP;{$ENDC}
002173      END;
002174
002175
002176  {$S SgABCdat}
002177      PROCEDURE TArrayScanner.Append(pRecord: Ptr);
002178      BEGIN
002179          {$IFC fTrace}BP(2);{$ENDC}
002180          TArray(SELF.collection).InsAt(SELF.position + 1, pRecord);
002181          SELF.position := SELF.position + 1;
002182          {$IFC fTrace}EP;{$ENDC}
002183      END;
002184
002185
002186  {$S SgABCdat}
002187      PROCEDURE TArrayScanner.Delete;
002188      BEGIN
002189          {$IFC fTrace}BP(2);{$ENDC}
002190          TArray(SELF.collection).DelAt(SELF.position);
002191          WITH SELF DO
002192              IF increment > 0 THEN
002193                  position := position - 1;
002194          {$IFC fTrace}EP;{$ENDC}
002195      END;
002196
002197
002198  {$S SgABCdat}
002199      PROCEDURE TArrayScanner.DeleteRest;
002200      BEGIN
002201          {$IFC fTrace}BP(2);{$ENDC}
002202          WITH SELF DO
002203              IF increment > 0 THEN
```

```
002204         {$H-}    TArray(collection).DelManyAt(position + 1, collection.size - position)
002205              ELSE
002206                  TArray(collection).DelManyAt(1, position - 1);  {$H+}
002207
002208          WITH SELF DO
002209              BEGIN
002210              collection.size := position;
002211              atEnd := TRUE;
002212              END;
002213          {$IFC fTrace}EP;{$ENDC}
002214      END;
002215
002216
002217  {$S SgABCdat}
002218      FUNCTION  TArrayScanner.Obtain: Ptr;
002219      BEGIN
002220          {$IFC fTrace}BP(1);{$ENDC}
002221          Obtain := TArray(SELF.collection).At(SELF.position);
002222          {$IFC fTrace}EP;{$ENDC}
002223      END;
002224
002225
002226  {$S SgABCdat}
002227      PROCEDURE TArrayScanner.Replace(pRecord: Ptr);
002228      BEGIN
002229          {$IFC fTrace}BP(2);{$ENDC}
002230          TArray(SELF.collection).PutAt(SELF.position, pRecord);
002231          {$IFC fTrace}EP;{$ENDC}
002232      END;
002233
002234
002235  {$S SgABCdat}
002236      FUNCTION  TArrayScanner.Scan(VAR pNextRecord: Ptr): BOOLEAN;
002237
002238          PROCEDURE AssignArrayScanVariable(anotherRecord: BOOLEAN);
002239          BEGIN
002240              IF anotherRecord THEN
002241                  pNextRecord := TArray(SELF.collection).At(SELF.position)
002242              ELSE
002243                  pNextRecord := NIL;
002244          END;
002245
002246      BEGIN
002247          {$IFC fTrace}BP(1);{$ENDC}
002248          Scan := SELF.Advance(AssignArrayScanVariable);
002249          {$IFC fTrace}EP;{$ENDC}
002250      END;
002251
```

```
002252
002253  {$S sInit1}
002254  BEGIN
002255
002256      availArrayScanner := NIL;
002257
002258  END;
002259  {$S SgCLAres}
002260
002261
002262  METHODS OF TStringScanner;
002263
002264
002265  {$S SgABCdat}
002266      FUNCTION  TStringScanner.CREATE(object: TObject; itsString: TString;
002267                                      itsInitialPosition: LONGINT; itsScanDirection: TScanDirection)
002268                                      : TStringScanner;
002269      BEGIN
002270          {$IFC fTrace}BP(1);{$ENDC}
002271          IF object = NIL THEN
002272              object := NewOrRecycledObject(mainHeap, THISCLASS, availStringScanner);
002273          SELF := TStringScanner(TScanner.CREATE(object, itsString, itsInitialPosition, itsScanDirection));
002274          SELF.actual := 0;
002275          {$IFC fTrace}EP;{$ENDC}
002276      END;
002277
002278
002279  {$S SgABCdat}
002280      PROCEDURE TStringScanner.Free;
002281      BEGIN
002282          {$IFC fTrace}BP(1);{$ENDC}
002283          RecycleObject(SELF, availStringScanner);
002284          {$IFC fTrace}EP;{$ENDC}
002285      END;
002286
002287
002288      {$IFC fDebugMethods}
002289      {$S SgCLAdbg}
002290      PROCEDURE TStringScanner.Fields(PROCEDURE Field(nameAndType: S255));
002291      BEGIN
002292          SUPERSELF.Fields(Field);
002293          Field('actual: LONGINT');
002294      END;
002295      {$S SgCLAres}
002296      {$ENDC}
002297
002298
002299  {$S SgABCdat}
```

```
002300      PROCEDURE TStringScanner.Append(character: CHAR);
002301      BEGIN
002302          {$IFC fTrace}BP(2);{$ENDC}
002303          TString(SELF.collection).InsAt(SELF.position + 1, character);
002304          SELF.position := SELF.position + 1;
002305          {$IFC fTrace}EP;{$ENDC}
002306      END;
002307
002308
002309  {$S SgABCdat}
002310      PROCEDURE TStringScanner.Delete;
002311      BEGIN
002312          {$IFC fTrace}BP(2);{$ENDC}
002313          TString(SELF.collection).DelAt(SELF.position);
002314          WITH SELF DO
002315              IF increment > 0 THEN
002316                  position := position - 1;
002317          {$IFC fTrace}EP;{$ENDC}
002318      END;
002319
002320
002321  {$S SgABCdat}
002322      PROCEDURE TStringScanner.DeleteRest;
002323      BEGIN
002324          {$IFC fTrace}BP(2);{$ENDC}
002325          WITH SELF DO
002326              IF increment > 0 THEN
002327         {$H-}    TString(collection).DelManyAt(position + 1, collection.size - position)
002328              ELSE
002329                  TString(collection).DelManyAt(1, position - 1);  {$H+}
002330
002331          WITH SELF DO
002332              BEGIN
002333              collection.size := position;
002334              atEnd := TRUE;
002335              END;
002336          {$IFC fTrace}EP;{$ENDC}
002337      END;
002338
002339
002340  {$S SgABCdat}
002341      FUNCTION  TStringScanner.Obtain: CHAR;
002342      BEGIN
002343          {$IFC fTrace}BP(1);{$ENDC}
002344          Obtain := TString(SELF.collection).At(SELF.position);
002345          {$IFC fTrace}EP;{$ENDC}
002346      END;
002347
```

```
002348
002349  {$S SgABCdat}
002350      PROCEDURE TStringScanner.Replace(character: CHAR);
002351      BEGIN
002352          {$IFC fTrace}BP(2);{$ENDC}
002353          TString(SELF.collection).PutAt(SELF.position, character);
002354          {$IFC fTrace}EP;{$ENDC}
002355      END;
002356
002357
002358  {$S SgABCdat}
002359      FUNCTION  TStringScanner.Scan(VAR nextChar: CHAR): BOOLEAN;
002360
002361          PROCEDURE AssignStringScanVariable(anotherChar: BOOLEAN);
002362          BEGIN
002363              IF anotherChar THEN
002364                  nextChar := TString(SELF.collection).At(SELF.position)
002365              ELSE
002366                  nextChar := CHAR(0);
002367          END;
002368
002369      BEGIN
002370          {$IFC fTrace}BP(1);{$ENDC}
002371          Scan := SELF.Advance(AssignStringScanVariable);
002372          {$IFC fTrace}EP;{$ENDC}
002373      END;
002374
002375
002376  {$S sResDat}
002377      FUNCTION  TStringScanner.ReadArray(heap: THeap; bytesPerRecord: INTEGER): TArray;
002378          VAR a:  TArray;
002379      BEGIN
002380          {$IFC fTrace}BP(2);{$ENDC}
002381          a := TArray.CREATE(NIL, heap, 0, bytesPerRecord);
002382          XferContiguous(xRead, a, 2, SELF);
002383          ReadArray := a;
002384          {$IFC fTrace}EP;{$ENDC}
002385      END;
002386
002387
002388  {$S sResDat}
002389      FUNCTION  TStringScanner.ReadNumber(numBytes: SizeOfNumber): LONGINT;
002390          VAR v:
002391              RECORD
002392              CASE INTEGER OF
002393                  1: (signExtension, short: INTEGER);
002394                  2: (long: LONGINT);
002395                  END;
```

```
002396      BEGIN
002397          {$IFC fTrace}BP(2);{$ENDC}
002398          v.long := 0;
002399          SELF.XferSequential(xRead, Ptr(ORD(@v)+4-numBytes), numBytes);
002400          IF numBytes=2 THEN
002401              IF v.short < 0 THEN
002402                  v.signExtension := -1;
002403          ReadNumber := v.long;
002404          {$IFC fTrace}EP;{$ENDC}
002405      END;
002406
002407
002408  {$S SgABCdat}
002409      FUNCTION  TStringScanner.ReadObject(heap: THeap): TObject;
002410          VAR class:  TClass;
002411              object: TObject;
002412      BEGIN
002413          {$IFC fTrace}BP(2);{$ENDC}
002414          class := TClass(SELF.ReadNumber(4));
002415          object := NewObject(heap, class);
002416          object.Read(SELF);
002417          ReadObject := object;
002418          {$IFC fTrace}EP;{$ENDC}
002419      END;
002420
002421
002422  {$S SgABCdat}
002423      PROCEDURE TStringScanner.WriteArray(a: TArray);
002424      BEGIN
002425          {$IFC fTrace}BP(2);{$ENDC}
002426          XferContiguous(xWrite, a, 2, SELF);
002427          {$IFC fTrace}EP;{$ENDC}
002428      END;
002429
002430
002431  {$S SgABCdat}
002432      PROCEDURE TStringScanner.WriteNumber(value: LONGINT; numBytes: SizeOfNumber);
002433      BEGIN
002434          {$IFC fTrace}BP(3);{$ENDC}
002435          SELF.XferSequential(xWrite, Ptr(ORD(@value)+4-numBytes), numBytes);
002436          {$IFC fTrace}EP;{$ENDC}
002437      END;
002438
002439
002440  {$S SgABCdat}
002441      PROCEDURE TStringScanner.WriteObject(object: TObject);
002442      BEGIN
002443          {$IFC fTrace}BP(2);{$ENDC}
```

```
002444          SELF.WriteNumber(ORD(object.Class), 4);
002445          object.Write(SELF);
002446          {$IFC fTrace}EP;{$ENDC}
002447      END;
002448
002449
002450  {$S SgABCdat}
002451      PROCEDURE TStringScanner.XferContiguous(whichWay: xReadWrite; collection: TCollection);
002452          VAR numToXfer:  INTEGER;
002453      BEGIN   {Transfer the size (as an INTEGER), class-specific fields, and members.
002454               Do not recur on the members.
002455               Do not transfer the class, the dynStart (=SizeOfClass), or the hole info (=zero).
002456               When reading, append the elements that are read.
002457               This only works for contiguous objects up to 32K members in size.}
002458          {$IFC fTrace}BP(3);{$ENDC}
002459          XferContiguous(whichWay, collection, 0, SELF);
002460          {$IFC fTrace}EP;{$ENDC}
002461      END;
002462
002463
002464  {$S SgABCdat}
002465      PROCEDURE TStringScanner.XferFields(whichWay: xReadWrite; object: TObject);
002466      BEGIN   {Transfers the bits of a TObject, excluding the class pointer and any dynamic part}
002467          {$IFC fTrace}BP(3);{$ENDC}
002468          SELF.XferSequential(whichWay,
002469                              Ptr(ORD(TH(object)^) + SIZEOF(TObject)),
002470                              SizeOfClass(object.Class) - SIZEOF(TObject));
002471          {$IFC fTrace}EP;{$ENDC}
002472      END;
002473
002474
002475  {$S SgABCdat}
002476      PROCEDURE TStringScanner.XferPString(whichWay: xReadWrite; pStr: TPString);
002477          VAR size: Byte;
002478      BEGIN
002479          {$IFC fTrace}BP(4);{$ENDC}
002480          IF whichWay = xWrite THEN
002481              size := Length(pStr^);
002482          SELF.XferSequential(whichWay, @size, 1);
002483          SELF.XferSequential(whichWay, Ptr(ORD(pStr)+1), size);
002484          {$IFC fTrace}EP;{$ENDC}
002485      END;
002486
002487
002488  {$S SgABCdat}
002489      PROCEDURE TStringScanner.XferRandom(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT;
002490                                          mode: TIOMode; offset: LONGINT);
002491      BEGIN
```

```
002492          {$IFC fTrace}BP(1);{$ENDC}
002493          CASE mode OF
002494              fAbsolute: SELF.Seek(offset);
002495              fRelative: SELF.Skip(offset);
002496              END;
002497          SELF.XferSequential(whichWay, pFirst, numBytes);
002498          {$IFC fTrace}EP;{$ENDC}
002499      END;
002500
002501
002502  {$S SgABCdat}
002503      PROCEDURE TStringScanner.XferSequential(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT);
002504      BEGIN
002505          {$IFC fTrace}BP(1);{$ENDC}
002506          WITH SELF, collection DO
002507              BEGIN
002508        {$H-} actual := Min(size - position, numBytes); {$H+}
002509        {$H-} collection.EditAt(size + 1, 0); {$H+}  {Maybe we should xfer in two steps instead}
002510              END;
002511          WITH SELF DO
002512              BEGIN
002513        {$H-} XferLeft(pFirst, Ptr(collection.AddrMember(position + 1)), actual); {$H+}
002514              position := position + actual;
002515              atEnd := position = collection.size;
002516              END;
002517          {$IFC fTrace}EP;{$ENDC}
002518      END;
002519
002520
002521  {$S sInit1}
002522  BEGIN
002523
002524      availStringScanner := NIL;
002525
002526  END;
002527  {$S SgCLAres}
002528
002529
002530  METHODS OF TFileScanner;
002531
002532
002533  {$S sResDat}
002534      FUNCTION  TFileScanner.CREATE(object: TObject; itsFile: TFile; manip: TAccesses): TFileScanner;
002535      BEGIN
002536          {$IFC fTrace}BP(5);{$ENDC}
002537          IF object = NIL THEN
002538              object := NewObject(itsFile.Heap, THISCLASS);
002539          SELF := TFileScanner(TScanner.CREATE(object, itsFile, 0, scanForward));
```

```
002540          SELF.actual := 0;
002541          SELF.accesses := manip;
002542          SELF.Open;
002543          TFile(SELF.collection).scanners.InsLast(SELF);
002544          {$IFC fTrace}EP;{$ENDC}
002545      END;
002546      {$S SgCLAres}
002547
002548
002549      {$IFC fDebugMethods}
002550      {$S SgCLAdbg}
002551      PROCEDURE TFileScanner.Fields(PROCEDURE Field(nameAndType: S255));
002552      BEGIN
002553          SUPERSELF.Fields(Field);
002554          Field('accesses: Byte');
002555          Field('refnum: INTEGER');
002556          Field('error: INTEGER');
002557      END;
002558      {$S SgCLAres}
002559      {$ENDC}
002560
002561
002562  {$S SgABCdat}
002563      PROCEDURE TFileScanner.FreeObject; {use FreeObject, rather than Free, so that we close the
002564                                          file if the user says fs.FreeObject (as in
002565                                          TDocManager.OpenSaved), as well as fs.Free}
002566      BEGIN
002567          {$IFC fTrace}BP(5);{$ENDC}
002568          SELF.Close;
002569          TFile(SELF.collection).scanners.DelObject(SELF, FALSE);
002570          SUPERSELF.FreeObject;
002571          {$IFC fTrace}EP;{$ENDC}
002572      END;
002573
002574
002575  {$S SgABCdat}
002576      PROCEDURE TFileScanner.Free;        {Free frees the TFile as well, if no other scanners still exist}
002577          VAR itsFile: TFile;
002578      BEGIN
002579          {$IFC fTrace}BP(5);{$ENDC}
002580          itsFile := TFile(SELF.collection);
002581          SELF.FreeObject;
002582          IF itsFile.scanners.size = 0 THEN
002583              itsFile.Free;
002584          {$IFC fTrace}EP;{$ENDC}
002585      END;
002586
002587
```

```
002588  {$S SgABCdat}
002589      PROCEDURE TFileScanner.Allocate(slack: LONGINT);
002590          VAR fsInfo:     FS_Info;
002591              pages:      LONGINT;
002592              actual:     LONGINT;
002593              newErr:     INTEGER;
002594      BEGIN
002595          {$IFC fTrace}BP(2);{$ENDC}
002596          Info(newErr, SELF.refnum, fsInfo);
002597          WITH fsInfo DO
002598              pages := ((size + slack + lpSize - 1) DIV lpSize) - ((pSize + lpSize - 1) DIV lpSize);
002599          IF pages > 0 THEN
002600              Allocate(newErr, SELF.refnum, TRUE, pages, actual);
002601          IF (newErr <= 0) AND (actual < pages) THEN
002602              Allocate(newErr, SELF.refnum, FALSE, pages - actual, actual);
002603    {$H-} LatestError(newErr, SELF.error); {$H+}
002604          {$IFC fTrace}EP;{$ENDC}
002605      END;
002606
002607
002608  {$S SgABCdat}
002609      PROCEDURE TFileScanner.Close;
002610          VAR newErr: INTEGER;
002611      BEGIN
002612          {$IFC fTrace}BP(2);{$ENDC}
002613          Close_Object(newErr, SELF.refnum);
002614    {$H-} LatestError(newErr, SELF.error); {$H+}
002615          {$IFC fTrace}EP;{$ENDC}
002616      END;
002617
002618
002619  {$S SgABCdat}
002620      PROCEDURE TFileScanner.Compact;
002621          VAR newErr:     INTEGER;
002622      BEGIN
002623          {$IFC fTrace}BP(2);{$ENDC}
002624          Compact(newErr, SELF.refnum);
002625    {$H-} LatestError(newErr, SELF.error); {$H+}
002626          {$IFC fTrace}EP;{$ENDC}
002627      END;
002628
002629
002630  {$S SgABCdat}
002631      PROCEDURE TFileScanner.Delete;
002632      BEGIN
002633          {$IFC fTrace}BP(2);{$ENDC}
002634          SELF.Skip(-1);
002635          {$IFC fTrace}EP;{$ENDC}
```

```
002636      END;
002637
002638
002639  {$S SgABCdat}
002640      PROCEDURE TFileScanner.DeleteRest;
002641          VAR newErr:     INTEGER;
002642      BEGIN
002643          {$IFC fTrace}BP(2);{$ENDC}
002644          Truncate(newErr, SELF.refnum);
002645    {$H-} LatestError(newErr, SELF.error); {$H+}
002646          WITH SELF DO
002647              BEGIN
002648              collection.size := position;
002649              atEnd := TRUE;
002650              END;
002651          {$IFC fTrace}EP;{$ENDC}
002652      END;
002653
002654
002655      PROCEDURE TFileScanner.Append(character: CHAR);
002656      BEGIN
002657          {$IFC fTrace}BP(2);{$ENDC}
002658          SELF.XferSequential(xWrite, Ptr(ORD(@character)+1), 1);
002659          {$IFC fTrace}EP;{$ENDC}
002660      END;
002661
002662
002663  {$S SgABCdat}
002664      FUNCTION  TFileScanner.Obtain: CHAR;
002665          VAR character: CHAR;
002666      BEGIN
002667          {$IFC fTrace}BP(1);{$ENDC}
002668          SELF.XferRandom(xRead, Ptr(ORD(@character) + 1), 1, fRelative, -1);
002669          Obtain := character;
002670          {$IFC fTrace}EP;{$ENDC}
002671      END;
002672
002673
002674  {$S sResDat}
002675  {$IFC LibraryVersion <= 20}
002676      PROCEDURE TFileScanner.Open;
002677          VAR pPath:      TPPathName;
002678              itsFile:    TFile;
002679      BEGIN
002680          {$IFC fTrace}BP(2);{$ENDC}
002681          itsFile := TFile(SELF.collection);
002682          pPath := @itsFile.path;
002683    {$H-} Open(SELF.error, pPath^, SELF.refnum, MSet(SELF.accesses)); {$H+}
```

```
002684          IF (SELF.error = 948) and (fWrite in SELF.accesses) then
002685              BEGIN
002686    {$H-}     Make_File(SELF.error, pPath^, 0);
002687              IF SELF.error <= 0 then
002688                  Open(SELF.error, pPath^, SELF.refnum, MSet(SELF.accesses)); {$H+}
002689              END;
002690          {$IFC fTrace}EP;{$ENDC}
002691      END;
002692
002693
002694  {$ELSEC}
002695      PROCEDURE TFileScanner.Open;
002696          VAR pPath:      TPPathName;
002697              itsFile:    TFile;
002698              pPass:      TPEName;
002699      BEGIN
002700          {$IFC fTrace}BP(2);{$ENDC}
002701          itsFile := TFile(SELF.collection);
002702          pPath := @itsFile.path;
002703          pPass := @itsFile.password;
002704    {$H-} Open_Secure(SELF.error, pPath^, SELF.refnum, MSet(SELF.accesses), pPass^); {$H+}
002705          IF (SELF.error = 948) and (fWrite in SELF.accesses) then
002706              BEGIN
002707    {$H-}     Make_Secure(SELF.error, pPath^, pPass^);
002708              IF SELF.error <= 0 then
002709                  Open_Secure(SELF.error, pPath^, SELF.refnum, MSet(SELF.accesses), pPass^); {$H+}
002710              END;
002711          {$IFC fTrace}EP;{$ENDC}
002712      END;
002713  {$ENDC}
002714
002715
002716  {$S SgABCdat}
002717      PROCEDURE TFileScanner.Replace(character: CHAR);
002718      BEGIN
002719          {$IFC fTrace}BP(2);{$ENDC}
002720          SELF.XferRandom(xWrite, Ptr(ORD(@character) + 1), 1, fRelative, -1);
002721          {$IFC fTrace}EP;{$ENDC}
002722      END;
002723
002724
002725  {$S SgABCdat}
002726      FUNCTION  TFileScanner.Scan(VAR nextChar: CHAR): BOOLEAN;
002727
002728          PROCEDURE AssignFileScanVariable(anotherChar: BOOLEAN);
002729          BEGIN
002730              IF anotherChar THEN
002731                  SELF.XferSequential(xRead, Ptr(ORD(@nextChar) + 1), 1)
```

```
002732              ELSE
002733                  nextChar := CHAR(0);
002734          END;
002735
002736      BEGIN
002737          {$IFC fTrace}BP(1);{$ENDC}
002738          Scan := SELF.Advance(AssignFileScanVariable);
002739          {$IFC fTrace}EP;{$ENDC}
002740      END;
002741
002742
002743  {$S sResDat}
002744      PROCEDURE TFileScanner.Seek(newPosition: LONGINT);
002745          VAR dummy: INTEGER;
002746      BEGIN
002747          {$IFC fTrace}BP(2);{$ENDC}
002748          SELF.XferRandom(xRead, @dummy, 0, fAbsolute, newPosition);
002749          {$IFC fTrace}EP;{$ENDC}
002750      END;
002751
002752
002753  {$S SgABCdat}
002754      PROCEDURE TFileScanner.Skip(deltaPos: LONGINT);
002755          VAR dummy: INTEGER;
002756      BEGIN
002757          {$IFC fTrace}BP(2);{$ENDC}
002758          SELF.XferRandom(xRead, @dummy, 0, fRelative, deltaPos);
002759          {$IFC fTrace}EP;{$ENDC}
002760      END;
002761
002762
002763  {$S sResDat}
002764      PROCEDURE TFileScanner.XferRandom(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT;
002765                                        mode: TIOMode; offset: LONGINT);
002766          VAR newErr:     INTEGER;
002767              osMode:     IOMode;
002768              fsInfo:     FS_Info;
002769              sched_err:  INTEGER;
002770      BEGIN
002771          {$IFC fTrace}BP(4);{$ENDC}
002772          osMode := IOMode(mode);
002773          WITH SELF DO {$H-}
002774              IF error <= 0 THEN
002775                  BEGIN
002776                  CASE whichWay OF
002777                      xRead:  BEGIN
002778                              Sched_Class(sched_err, FALSE);
002779                              Read_Data(newErr, refnum, ord(pFirst), numBytes, actual, osMode, offset);
```

```
002780                          Sched_Class(sched_err, TRUE);
002781                          END;
002782                      xWrite: BEGIN
002783                          Sched_Class(sched_err, FALSE);
002784                          Write_Data(newErr, refnum, ord(pFirst), numBytes, actual, osMode, offset);
002785                          Sched_Class(sched_err, TRUE);
002786                          collection.size := Max(position + actual, collection.size);
002787                          END;
002788                      END;
002789
002790                  IF (newErr = 956) OR (newErr = 963) OR (newErr = 883) OR (newErr = 882) OR
002791                     (newErr = 848) THEN {EOF}
002792                      newErr := 0;
002793
002794                  IF mode = fSequential THEN  {do it fast}
002795                      position := position + actual
002796                  ELSE                        {play it safe}
002797                      BEGIN
002798                      Info(newErr, refnum, fsInfo);
002799                      position := fsInfo.fMark;
002800                      collection.size := fsInfo.size;
002801                      END;
002802
002803                  atEnd := position = collection.size;
002804                  LatestError(newErr, error); {$H+}
002805                  END;
002806          {$IFC fTrace}EP;{$ENDC}
002807      END;
002808
002809
002810  {$S sResDat}
002811      PROCEDURE TFileScanner.XferSequential(whichWay: xReadWrite; pFirst: Ptr; numBytes: LONGINT);
002812      BEGIN
002813          {$IFC fTrace}BP(1);{$ENDC}
002814          SELF.XferRandom(whichWay, pFirst, numBytes, fSequential, 0);
002815          {$IFC fTrace}EP;{$ENDC}
002816      END;
002817
002818
002819  {$S sInit1}
002820  END;
002821  {$S SgCLAres}
002822
002823
002824  {$IFC compatibleLists} {Backward Compatibility}
002825  METHODS OF TDynamicArray;
002826
002827
```

```
002828      FUNCTION  TDynamicArray.CREATE(object: TObject; heap: THeap; bytesPerRecord: INTEGER;
002829                                      initialSize: INTEGER): TDynamicArray;
002830      BEGIN
002831          {$IFC fTrace}BP(1);{$ENDC}
002832          IF ODD(bytesPerRecord) THEN
002833              bytesPerRecord := bytesPerRecord + 1;
002834          SELF := POINTER(ORD(TArray.CREATE(object, heap, initialSize, bytesPerRecord))); {NB reversed args}
002835          Handle(SELF)^^ := ORD(THISCLASS);
002836          SELF.EditAt(1, initialSize);
002837          {$IFC fTrace}EP;{$ENDC}
002838      END;
002839
002840
002841      PROCEDURE TDynamicArray.BeSize(newSize: INTEGER);
002842      BEGIN
002843          SELF.EditAt(SELF.size + 1, newSize - SELF.size);
002844      END;
002845
002846
002847      FUNCTION  TDynamicArray.Class: TClass; {So New- & Resize- DynObject will use correct object size}
002848      BEGIN
002849          Class := cArray;
002850      END;
002851
002852
002853      FUNCTION  TDynamicArray.numRecords: INTEGER;
002854      BEGIN
002855          numRecords := SELF.size;
002856      END;
002857
002858
002859  {$S sInit1}
002860  END;
002861  {$S SgCLAres}
002862
002863
002864  METHODS OF TIndexList;
002865
002866
002867      FUNCTION  TIndexList.CREATE(object: TObject; heap: THeap; initialSize: INTEGER): TIndexList;
002868      BEGIN
002869          {$IFC fTrace}BP(1);{$ENDC}
002870          SELF := POINTER(ORD(TList.CREATE(object, heap, initialSize)));
002871          Handle(SELF)^^ := ORD(THISCLASS);
002872          SELF.EditAt(1, initialSize);
002873          {$IFC fTrace}EP;{$ENDC}
002874      END;
002875
```

```
002876
002877      FUNCTION  TIndexList.Class: TClass; {So New- & Resize- DynObject will use correct object size}
002878      BEGIN
002879          Class := cList;
002880      END;
002881
002882
002883      FUNCTION  TIndexList.numElements: INTEGER;
002884      BEGIN
002885          numElements := SELF.size;
002886      END;
002887
002888
002889  {$S sInit1}
002890  END;
002891  {$S SgCLAres}
002892
002893
002894  METHODS OF TLinkList;
002895
002896
002897      FUNCTION  TLinkList.CREATE(object: TObject; heap: THeap): TLinkList;
002898      BEGIN
002899          {$IFC fTrace}BP(1);{$ENDC}
002900          SELF := POINTER(ORD(TList.CREATE(object, heap, 0)));
002901          Handle(SELF)^^ := ORD(THISCLASS);
002902          {$IFC fTrace}EP;{$ENDC}
002903      END;
002904
002905
002906      FUNCTION  TLinkList.numElements: INTEGER;
002907      BEGIN
002908          numElements := SELF.size;
002909      END;
002910
002911
002912  {$S sInit1}
002913  END;
002914  {$S SgCLAres}
002915
002916
002917  METHODS OF TBlockList;
002918
002919
002920      FUNCTION  TBlockList.CREATE(object: TObject; heap: THeap; itsMinBlockSize: INTEGER): TBlockList;
002921      BEGIN
002922          {$IFC fTrace}BP(1);{$ENDC}
002923          SELF := POINTER(ORD(TList.CREATE(object, heap, 0)));
```

```
002924          Handle(SELF)^^ := ORD(THISCLASS);
002925          {$IFC fTrace}EP;{$ENDC}
002926      END;
002927
002928
002929      FUNCTION  TBlockList.numElements: INTEGER;
002930      BEGIN
002931          numElements := SELF.size;
002932      END;
002933
002934
002935  {$S sInit1}
002936  END;
002937  {$S SgCLAres}
002938
002939
002940  METHODS OF TFileStream;
002941
002942
002943      FUNCTION  TFileStream.CREATE(object: TObject; heap: THeap; path: S255; manip: TAccesses): TFileStream;
002944      BEGIN
002945          {$IFC fTrace}BP(1);{$ENDC}
002946          IF object = NIL THEN
002947              object := NewObject(heap, THISCLASS);
002948          SELF := TFileStream(TFileScanner.CREATE(object, TFile.CREATE(NIL, heap, path, ''), manip));
002949          {$IFC fTrace}EP;{$ENDC}
002950      END;
002951
002952
002953      FUNCTION  TFileStream.Size: LONGINT;
002954      BEGIN
002955          {$IFC fTrace}BP(1);{$ENDC}
002956          Size := SELF.collection.size;
002957          {$IFC fTrace}EP;{$ENDC}
002958      END;
002959
002960
002961  {$S sInit1}
002962  END;
002963  {$S SgCLAres}
002964
002965
002966  {$S SgCLAcld}
002967  PROCEDURE FileDelete(path: S255);
002968      VAR osPath:     Pathname;
002969          error:      INTEGER;
002970  BEGIN
002971      osPath := path;         { THIS IS THE SECOND TIME WE COPY THE STRING !!!! }
```

```
002972      Kill_Object(error, osPath);
002973  END;
002974  {$S SgCLAres}
002975
002976
002977  {$S SgCLAcld}
002978  PROCEDURE FileLookup(VAR error: INTEGER; path: S255);
002979      VAR refInfo:    FS_Info;
002980          osPath:     Pathname;
002981  BEGIN
002982      osPath := path;         { THIS IS THE SECOND TIME WE COPY THE STRING !!!! }
002983      Lookup(error, osPath, refInfo);
002984  END;
002985  {$S SgCLAres}
002986
002987
002988  {$S SgCLAcld}
002989  PROCEDURE FileRename(oldPath, newPath: S255);
002990      VAR osPath:         Pathname;
002991          osEname:        E_Name;
002992          error:          INTEGER;
002993          centerHyphen:   INTEGER;
002994  BEGIN
002995      osPath := oldPath;
002996      centerHyphen := pos('-{', newPath);
002997      osEname := copy(newPath, centerHyphen+1, length(newPath)-centerHyphen);
002998      Rename_Entry(error, osPath, osEname);
002999  END;
003000  {$S SgCLAres}
003001
003002
003003  {$S SgCLAcld}
003004  FUNCTION FileModified(path: S255): LONGINT;
003005      VAR refInfo:    FS_Info;
003006          osPath:     Pathname;
003007          error:      INTEGER;
003008  BEGIN
003009      osPath := path;         { THIS IS THE SECOND TIME WE COPY THE STRING !!!! }
003010      Lookup(error, osPath, refInfo);
003011      IF error <= 0 THEN
003012          FileModified := refInfo.DTM
003013      ELSE
003014          FileModified := -1;
003015  END;
003016  {$S SgCLAres}
003017
003018
003019  {$ENDC} {Backward Compatibility}
```

```
003020
003021
003022
003023

End of File -- Lines: 3023 Characters: 79889
```

```
================================================================================================
FILE: "LIBTK/UOBJECT4.TEXT"
================================================================================================

000001  {INCLUDE FILE UOBJECT4 -- KITBUG}
000002  {Copyright 1983, 1984, Apple Computer, Inc.}
000003
000004  {changed 04/30 1412 In GetDollarD make sure the constants we are searching for don't appear in the
000005                          body of the procedure}
000006
000007  { ==================================== VALIDITY CHECKS ==================================== }
000008
000009
000010  {$S SgCLAcld}
000011
000012
000013  FUNCTION  ValidGlobalAddress(addr: LONGINT): BOOLEAN;
000014  BEGIN
000015          {$IFC fMaxTrace}BP(1);{$ENDC}
000016          {$IFC fMaxTrace}EP;{$ENDC}
000017      ValidGlobalAddress := (%_GetA5 > addr) AND (addr > ORD(@addr));
000018  END;
000019
000020
000021  FUNCTION  ValidSTP(stAddr: LONGINT): BOOLEAN;
000022      VAR count:  INTEGER;
000023          hiWord: INTEGER;
000024  BEGIN
000025          {$IFC fMaxTrace}BP(1);{$ENDC}
000026          {$IFC fMaxTrace}EP;{$ENDC}
000027      IF ValidGlobalAddress(stAddr) AND ValidGlobalAddress(stAddr+3) AND NOT ODD(stAddr) THEN
000028          BEGIN
000029          count := 100;     {Prevent infinite loops}
000030          hiWord := 0;
000031          WHILE ValidGlobalAddress(stAddr-4) AND ValidGlobalAddress(stAddr-1) AND NOT ODD(stAddr) AND
000032                (count > 0) DO
000033              BEGIN
000034        {$R-} hiWord := TpINTEGER(stAddr-4)^;
000035              stAddr := TpLONGINT(stAddr-4)^; {$IFC fRngObject}{$R+}{$ENDC}
000036              count := count - 1;
000037              END;
000038          ValidSTP := hiWord = -1;
000039          END
000040      ELSE
000041          ValidSTP := FALSE;
000042  END;
000043
```

```
000044
000045  FUNCTION  ValidObject(hndl: Handle): BOOLEAN;
000046  BEGIN
000047          {$IFC fMaxTrace}BP(1);{$ENDC}
000048          {$IFC fMaxTrace}EP;{$ENDC}
000049      ValidObject := FALSE;
000050      IF (hndl <> NIL) AND (cObject <> NIL) THEN                      {Not NIL; and we have made a heap}
000051          IF ValidDataAddress(ORD(hndl)) THEN                         {Handle reasonable}
000052              IF ValidDataAddress(ORD(hndl^)) THEN                    {Master ptr reasonable}
000053                  IF ValidSTP(ORD(ClassPtr(hndl))) THEN               {Reasonable stp}
000054                      ValidObject := TRUE;                            {Go for it}
000055  END;
000056
000057
000058  { ================================== GARBAGE COLLECTOR ================================== }
000059
000060
000061  {$S SgCLAcld}
000062
000063
000064  PROCEDURE MarkHeap{(heap: THeap; mpAddress: LONGINT)};
000065
000066  { MarkHeap accepts two parameters: (1) a pointer (heap) to the document heap and (2) the address, }
000067  { (mpAddress) of a "root" master pointer from which all other accessible objects on heap can be reached. }
000068
000069  { MarkHeap marks all objects that are "in-use" by marking the root object, all objects that the root object }
000070  { has a handle on, all objects that those objects have handles on, etc.  Marking is accomplished by setting }
000071  { the high order bit (bit 31) of the master pointer that points to the object which is to be marked. }
000072
000073  { Although MarkHeap operates depth-first, it is NOT recursive.  Thus, it can mark long chains of objects }
000074  { without causing stack expansion.  If w.e => x, x.f => y and y.g => z, then while y is being scanned, }
000075  { x.f => w.e.  Thus, when y returns to x for further marking starting after f, x can know where it will }
000076  { have to return to when its scan is complete.  The comments below assume that the scan has reached y.g }
000077
000078      TYPE TOffsets = RECORD
000079              objectOffset:   INTEGER;    { x - mpFirst: where the object's master ptr is in the heap }
000080              fieldOffset:    INTEGER;    { @x.f - @x^^: where the field is in the object }
000081          END;
000082
000083      VAR hz: THz;                    { heap as a UnitHz type }
000084          mpFirst: LONGINT;           { The address of the first master pointer in the heap }
000085          mpLast: LONGINT;            { The address of the last master pointer in the heap }
000086
000087          blockPtr: TBk;              { A pointer to the first (size) word of the storage block of y}
000088          sizeInWords: INTEGER;       { The size found there }
000089
000090          firstFieldAddress: LONGINT; { @y^^, the address of y's first data field (usually a method-table ptr)}
000091          lastFieldAddress: LONGINT;  { The upper limit of the fieldAddress loop--the last 4-byte field of y }
```

```
000092
000093          mpPtr: TpLONGINT;           { A handle as a pointer to a LONGINT (the master pointer) }
000094          mp: LONGINT;                { The master pointer value, z^, i.e., the object data address }
000095
000096          fieldOffset: INTEGER;       { @y.g - @y^^ }
000097          fieldAddress: LONGINT;      { The address of the field y.g, which may or may not be a handle.
000098                                          It increases by twos because a handle can start on any even address}
000099          previous: TOffsets;         { Two offsets representing @x.f: x - mpFirst & @x.f - @x^^.
000100                                        A pointer to x.f will be stashed there while z is scanned,
000101                                        in the form of two offsets (see "previous") }
000102
000103          hndlAddress: LONGINT;       { The handle z found in or to be replaced in y.g }
000104
000105          goodHandleFound: BOOLEAN;   { TRUE if a handle to an unmarked object was found in the fields of
000106                                        the present object; otherwise, FALSE. }
000107
000108  BEGIN   { MarkHeap }
000109          {$IFC fMaxTrace}BP(1);{$ENDC}
000110          {$IFC fMaxTrace}EP;{$ENDC}
000111
000112      hz := THz(heap);           { A pointer to the heap }
000113      mpFirst := ORD(@hz^.argpPool);                    { The address of the first master pointer }
000114      mpLast := mpFirst + (4 * (hz^.ipPoolMac - 1));    { The address of the last master pointer }
000115
000116      fieldOffset := 0;                     { The offset from firstFieldAddress of the first field to
000117                                              consider }
000118      goodHandleFound := TRUE;
000119      previous.objectOffset := 1;         { An illegal value to flag the end of the entire marking operation }
000120
000121  {$IFC LibraryVersion > 20}
000122      {Mark the hrgpnob field of the Hz}
000123      mpPtr := TpLONGINT(hz^.hrgpnob);
000124      mpPtr^ := mpPtr^ + $80000000;
000125
000126      {Mark the hScramble field of the Hz}
000127      mpPtr := TpLONGINT(hz^.hScramble);
000128      mpPtr^ := mpPtr^ + $80000000;
000129  {$ENDC}
000130
000131      mpPtr := TpLONGINT(mpAddress);        { The handle of y }
000132      mpPtr^ := mpPtr^ + $80000000;       { Mark the master pointer which points to the present object }
000133
000134      REPEAT                              { Loop through all accessible objects}
000135
000136          firstFieldAddress := mpPtr^;                  { The address of the first field of y }
000137          blockPtr := TBk(firstFieldAddress - 4);       { The size word of the header of the object }
000138          sizeInWords := blockPtr^.hdr.cw;              { The size of the object, in words }
000139          lastFieldAddress := firstFieldAddress + sizeInWords + sizeInWords - 6; { The last 4-byte field }
```

```
000140
000141          fieldAddress := firstFieldAddress + fieldOffset;    { Where to start or resume the scan of y  }
000142
000143          IF (NOT goodHandleFound) THEN
000144              BEGIN    { We have just returned to y after scanning z }
000145              previous := TOffsets(TpLONGINT(fieldAddress)^);{ Restore previous offsets from field y.g }
000146              TpLONGINT(fieldAddress)^ := hndlAddress;       { Restore the original contents of y.g, which
000147                                                              was z }
000148
000149              fieldAddress := fieldAddress + 2;           { Advance to the next potential handle }
000150              fieldOffset := fieldOffset + 2;
000151              END;
000152
000153          goodHandleFound := FALSE;    { No handle to an unmarked object has been found yet }
000154
000155             { Scan the fields of the present object in search of a handle to an unmarked object }
000156          WHILE ((fieldAddress <= lastFieldAddress) AND (NOT goodHandleFound)) DO
000157              BEGIN
000158              hndlAddress := TpLONGINT(fieldAddress)^;       { Get what may be the address of a master pointer }
000159
000160              IF (hndlAddress >= mpFirst) THEN
000161                  IF (hndlAddress <= mpLast) THEN
000162                      IF (LIntAndLInt(hndlAddress - mpFirst, 3) = 0) THEN
000163                          BEGIN
000164                                      { if the address of the alleged master pointer lies between the }
000165                                      { addresses of the first and last master pointers, inclusive, and if }
000166                                      { the address of the alleged master pointer lies a multiple of 4 bytes }
000167                                      { (the length of a master pointer) from the address of the first }
000168                                      { master pointer, then the given address is the address of a master }
000169                                      { pointer (i.e. it is a valid handle).}
000170                          mpPtr := TpLONGINT(hndlAddress);  { Get a handle on the validated master pointer }
000171                          mp := ORD(mpPtr^);
000172                          IF (mp >= 0) THEN { unmarked }
000173                              IF NOT (((mp >= mpFirst) AND (mp <= mpLast)) OR (mp = 1)) THEN
000174                                  BEGIN  { not on the free list; it must be in the heap proper }
000175                                  goodHandleFound := TRUE;    { A handle to an unmarked object has been found }
000176
000177                                  TOffsets(TpLONGINT(fieldAddress)^) := previous; { Save offsets in the
000178                                                                                    field y.g }
000179
000180                                  previous.fieldOffset := fieldOffset;  { y's current offsets are z's
000181                                                                          previous ones }
000182                                  previous.objectOffset := mpAddress - mpFirst;
000183
000184                                  mpAddress := hndlAddress;       { The handle of z }
000185                                  END;
000186                          END;
000187
```

```
000188              fieldAddress := fieldAddress + 2;   { Advance to the next potential handle }
000189              fieldOffset := fieldOffset + 2;     { Set offset to next potential handle }
000190              END;
000191
000192          IF goodHandleFound THEN             { y.g contained the handle of z }
000193              BEGIN
000194              mpPtr^ := mpPtr^ + $80000000;   { Mark the master pointer of z }
000195              fieldOffset := 0;               { Prepare to scan z }
000196              END
000197          ELSE
000198              BEGIN    { Finished examining the fields of y.  Prepare to return to x.f }
000199              hndlAddress := mpAddress;       { The handle y will be put back into x.f where it belongs }
000200              fieldOffset := previous.fieldOffset;            { Restore fieldOffset to @x.f - @x^^ }
000201              mpAddress := mpFirst + previous.objectOffset;   { Restore mpAddress to x }
000202              mpPtr := TpLONGINT(mpAddress);                    { The handle of y }
000203              END;
000204
000205        UNTIL previous.objectOffset = 1;    { until all the fields of the original object have been examined }
000206  END;   { MarkHeap }
000207
000208
000209  PROCEDURE SweepHeap{(heap: THeap; report: BOOLEAN)};
000210
000211  { This procedure sweeps through all existing objects on the document heap specified by the handle heap.  }
000212  { If the parameter report has the value TRUE, then the classes of all unmarked objects are displayed on the }
000213  { alternate screen; otherwise, if report is FALSE, the unmarked objects are quietly freed-up. }
000214
000215      VAR tempPtr: TpLONGINT; { a temporary pointer used either to carry out simple indirection or to mark a }
000216                           { master pointer }
000217
000218      PROCEDURE CollectGarbage (obj: TObject);
000219
000220      { This procedure accepts a handle, obj, to an object and frees or reports that object (depending on the }
000221      { value of SweepHeap's parameter, report) if its master pointer is not marked.  If, the }
000222      { object's master pointer is marked, then this procedure unmarks the object's master pointer but }
000223      { otherwise leaves the object alone. }
000224
000225          VAR mpAddress:  LONGINT;    { the address of the master pointer specified by the handle obj }
000226              clsName:    TClassName; { the name of that class }
000227              hexOrd:     S8;         { the handle of the object, as a hex string }
000228
000229      BEGIN    { CollectGarbage }
000230
000231          mpAddress := ORD(obj);
000232
000233          tempPtr := TpLONGINT(obj);   { get a handle of the right type on the given object OBJ }
000234          IF (tempPtr^ < 0) THEN
000235              BEGIN    { if the given object OBJ is marked }
```

```
000236              tempPtr := TpLONGINT(mpAddress);   { Unmark the master pointer that points to the present object }
000237              tempPtr^ := tempPtr^ - $80000000;  { Note: 2^31 = $80000000 }
000238              END
000239          ELSE IF report THEN
000240              BEGIN
000241              WriteLn;
000242              IF ValidObject(Handle(obj)) THEN
000243                  CpToCn(TPSliceTable(ClassPtr(Handle(obj))), TS8(clsName))
000244              ELSE
000245                  clsName := '????????';
000246
000247              LIntToHex(ORD(obj), @hexOrd);
000248              Write (CHR(7), 'Found garbage object $', hexOrd, ' of class ', clsName);  { Report the garbage }
000249              END
000250          ELSE
000251              FreeH(THz(heap), TH(obj));  { It is unmarked, i.e., garbage.  Free it. }
000252
000253      END;    { CollectGarbage }
000254
000255  BEGIN    { SweepHeap }
000256          {$IFC fMaxTrace}BP(1);{$ENDC}
000257          {$IFC fMaxTrace}EP;{$ENDC}
000258      EachObject(heap, CollectGarbage);
000259  END;     { SweepHeap }
000260
000261
000262  { ================================== ABCBREAK ================================== }
000263
000264
000265  {$IFC fDbgObject}
000266  PROCEDURE TallyZero; FORWARD;
000267  {$ENDC}
000268
000269
000270  {$S sError}
000271  PROCEDURE ABCBreak{(s: S255; errCode: LONGINT)};
000272      VAR asHex: S8;
000273  BEGIN
000274      {$IFC fDbgObject}
000275      WriteLn;
000276      Write(CHR(7), s);       {Beep}
000277      IF errCode <> 0 THEN
000278          BEGIN
000279          LIntToHex(errCode, @asHex);
000280          Write(': ', errCode:1, ' = $', asHex);
000281          END;
000282      WriteLn;
000283      {Turn off all tracing, tallying, etc.}
```

```
000284      tallyingCalls := FALSE;
000285      TallyZero;
000286      fTraceEnabled := FALSE;
000287      defTraceCount := 0;
000288      traceCount := defTraceCount;
000289      returnToMain := TRUE;
000290      EntDebugger(' ', 'Error caused ABCBreak call');
000291      {$ELSEC}
000292      HALT;
000293      {$ENDC}
000294  END;
000295  {$S SgCLAcld}
000296
000297
000298  { ===================================== $D DECODING ====================================== }
000299
000300
000301  {$IFC fTrace OR fDebugMethods}
000302
000303  {$S SgCLAdbg}
000304
000305
000306  FUNCTION GetDollarD(pFrame: TppINTEGER;
000307                      VAR nameOfClass: TClassName; VAR nameOfMethod: S8; VAR nextPC: LONGINT): BOOLEAN;
000308      LABEL 1;
000309
000310      VAR pname:              TPByte;
000311          pPC:                TppINTEGER;
000312          pc:                 TpINTEGER;
000313          startOfSegment:     TpLONGINT;
000314          endOfSegment:       TpINTEGER;
000315          pcl:                TpLONGINT;
000316          fBothClassAndProc:  BOOLEAN;
000317
000318      PROCEDURE SwapIn(valueString: S8);
000319          BEGIN
000320          END;
000321
000322      PROCEDURE CopyName(VAR anyName: S8);
000323          VAR j: INTEGER;
000324      BEGIN
000325          anyName := '12345678';
000326          FOR j := 1 TO 8 DO
000327              BEGIN
000328              anyName[j] := CHR(Wand(pname^, 127));
000329              pname := TPByte(ORD(pname)+1);
000330              END;
000331      END;
```

```
000332
000333      PROCEDURE AdvancePC;
000334
000335      BEGIN
000336          IF ORD(pc) >= ORD(endOfSegment) THEN
000337              GOTO 1;
000338          pc := TpInteger(ORD(pc)+2);
000339      END;
000340
000341  BEGIN
000342          {$IFC fMaxTrace}BP(1);{$ENDC}
000343          {$IFC fMaxTrace}EP;{$ENDC}
000344      pPC := TppINTEGER(ORD(pFrame) + 4);
000345      pc := pPC^;
000346      nameOfClass := '';
000347      nameOfMethod := '';
000348      nextPC := 0;
000349      GetDollarD := FALSE;
000350
000351      IF ORD(pc) <> 0 THEN
000352          BEGIN
000353      {$R-} SwapIn(TPS8(pc)^); {$IFC fRngObject} {$R+} {$ENDC} {Be sure the code is swapped in}
000354
000355         startOfSegment := TpLONGINT(LIntAndLint(LONGINT(PC), $FFFE0000));
000356         endOfSegment := TpINTEGER(LONGINT(startOfSegment) + LIntAndLint(startOfSegment^, $00FFFFFF) {length} );
000357
000358              {We add the -1 to the following tests so that the things we are searching for don't
000359                  appear in the body of the procedure.}
000360          WHILE (pc^-1) <> ($4E5E-1) DO                   {search for UNLK A6}
000361              IF ORD(pc) >= ORD(endOfSegment) THEN
000362                  GOTO 1
000363              ELSE
000364                  pc := TpINTEGER(ORD(pc)+2);
000365          WHILE ((pc^-1) <> ($4E75-1)) AND
000366                ((pc^-1) <> ($4ED0-1)) DO                 {search for RTS or JMP (A0)}
000367              IF ORD(pc) >= ORD(endOfSegment) THEN
000368                  GOTO 1
000369              ELSE
000370                  pc := TpINTEGER(ORD(pc)+2);
000371
000372          nextPC := ORD(pc);
000373          GetDollarD := TRUE;
000374
000375          pname := TPByte(ORD(pc)+3);
000376          fBothClassAndProc := pname^ < 0;
000377          pname := TPByte(ORD(pname)-1);
000378          CopyName(nameOfMethod);
000379          IF fBothClassAndProc THEN
```

```
000380              CopyName(S8(nameOfClass))
000381          ELSE
000382              nameOfClass := '';
000383          END;
000384  1:
000385  END;
000386  {$ENDC}
000387
000388
000389  { ================================== CALL TALLY ================================== }
000390
000391
000392  {$IFC fTrace}
000393
000394
000395  {$S SgCLAini}       {  *** NB ***  Is this Sg necessary?  }
000396
000397
000398  PROCEDURE TallyStart;
000399      VAR timeNow:    LONGINT;
000400          i:          INTEGER;
000401          arrSize:    INTEGER;
000402          elapsed:    LONGINT; (*^*)
000403  BEGIN
000404      IF tallies = NIL THEN
000405          BEGIN
000406          {array size must be <= maxTallies; imposed by declaration of tallies global variable.}
000407          arrSize := Min(numMethods, maxTallies);
000408          tallies := THTallies(TArray.CREATE(NIL, mainHeap, arrSize, SIZEOF(TTally)));
000409          TArray(tallies).InsNullsAt(1, arrSize);
000410          elapsed := 0;(*^*)
000411          END
000412      ELSE {continuing}
000413          elapsed := stopTime - startTime;(*^*)
000414
000415      timeNow := MicroTimer;
000416      startTime := timeNow - elapsed; (*^*)
000417      FOR i := 0 TO tabLevel DO    {BP's already passed}
000418          traceTimes[i] := timeNow; (*^*)
000419      stopTime := timeNow;
000420
000421      tallyingCalls := TRUE;
000422  END;
000423
000424
000425  PROCEDURE TallyZero;
000426  BEGIN
000427      IF tallies <> NIL THEN
```

```
000428          BEGIN
000429          Free(TArray(tallies));
000430          tallies := NIL;
000431          END;
000432  END;
000433
000434
000435  FUNCTION TallySlot(pc: LONGINT): INTEGER;
000436
000437      FUNCTION ComparePC(hashIndex: INTEGER): THashCompare;
000438          VAR myPC:   LONGINT;
000439      BEGIN
000440          myPC := tallies^^.recs[hashIndex].epPC;
000441          IF myPC = 0 THEN
000442              ComparePC := cHole
000443          ELSE
000444          IF myPC = pc THEN
000445              ComparePC := cMatch
000446          ELSE
000447              ComparePC := cMismatch;
000448      END;
000449
000450  BEGIN
000451      TallySlot := LookupInHashArray(tallies^^.header.size, pc, FALSE, ComparePC);
000452  END;
000453
000454
000455  PROCEDURE Tally(pc, micSecs: LONGINT);
000456      VAR slot:       INTEGER;
000457          segNum:     INTEGER;
000458          pPC:        TpByte;
000459  BEGIN
000460      pPC := TPByte(@pc);
000461      pPC^ := 0;  {occasionally, a return addr hibyte is nonzero! no one knows why...}
000462
000463      slot := TallySlot(pc);
000464      WITH tallies^^.recs[ABS(slot)] DO
000465          IF slot > 0 THEN
000466              BEGIN
000467              count := count + 1;
000468              microseconds := microseconds + micsecs;
000469              END
000470          ELSE
000471          IF slot < 0 THEN
000472              BEGIN
000473              segNum := TpINTEGER(pPC)^ DIV 2;
000474              IF segNum = 0 THEN
000475                  ABCBreak('Impossible Tally PC', pc)
```

```
000476              ELSE
000477                  BEGIN
000478                  epPC := pc;
000479                  count:= 1;
000480                  microseconds := micSecs;
000481                  END;
000482              END
000483          ELSE
000484              BEGIN
000485              ABCBreak('Tally table full--more non-method procedures had EP''s than expected',
000486                      tallies^^.header.size);
000487              tallyingCalls := FALSE;
000488              END;
000489  END;
000490
000491
000492  PROCEDURE TallyReport;
000493      VAR totalCalls:     LONGINT;
000494          totalTime:      LONGINT;
000495          callees:        INTEGER;
000496          slot:           INTEGER;
000497          calls:          INTEGER;
000498          micSecs:        LONGINT;
000499          roundoff:       LONGINT;
000500          i:              INTEGER;
000501          j:              INTEGER;
000502          sortKeys:       THIdxArray;
000503          segCount:       ARRAY [0..127] OF LONGINT;
000504          segTime:        ARRAY [0..127] OF LONGINT;
000505          pc:             LONGINT;
000506          sortBy:         INTEGER;
000507          swapem:         BOOLEAN;
000508          sloti:          INTEGER;
000509          slotj:          INTEGER;
000510          pctg:           INTEGER;
000511          elapsed:        LONGINT;
000512          segName:        S8;
000513          segNum:         INTEGER;
000514          cState:         TConvResult;
000515          wantCalled:     BOOLEAN;
000516          clsName:        TClassName;
000517          mthName:        S8;
000518          nextPC:         LONGINT;
000519          inStr:          S255;
000520          hexPC:          S8;
000521
000522          PROCEDURE ReadSegNames;
000523              CONST
```

```
000524                  bSegTable = $9A;
000525                  bEOFMark = $00;
000526                  bModuleName = $80;
000527                  bCodeBlock = $85;
000528
000529                  modNameSkip = 8; {# bytes to skip in module name block, to get segment name}
000530
000531                  allBlanks = '        '; {8 blanks}
000532                  blankSeg  = 'BLANKSEG';
000533
000534              TYPE
000535                  SegTblEntry = RECORD
000536                      SegName:        PACKED ARRAY[1..8] OF CHAR;
000537                      SegNumber:      INTEGER;
000538                      Version1:       LONGINT;
000539                      Version2:       LONGINT;
000540                      END;
000541
000542              VAR prcsInfo:       ProcInfoRec;
000543                  error:          INTEGER;
000544                  aFile:          TFile;
000545                  scanner:        TFileScanner;
000546                  blkType:        LONGINT;
000547                  blkSize:        LONGINT;
000548                  nSegments:      LONGINT;
000549                  segblk:         SegTblEntry;
000550                  addr:           LONGINT;
000551                  i:              INTEGER;
000552
000553          BEGIN
000554              Info_Process(error, My_id, prcsInfo);
000555              IF error <= 0 THEN
000556                  BEGIN
000557                  segName := allBlanks;
000558                  segNames := TArray.CREATE(NIL, mainHeap, 127, SIZEOF(S8));
000559                  segNames.InsNullsAt(1, 127);
000560
000561                  aFile := TFile.CREATE(NIL, mainHeap, prcsInfo.progPathName, '');
000562                  scanner := TFileScanner.CREATE(NIL, aFile, [fRead]);
000563                  WriteLn('Reading segment numbers and names from ', prcsInfo.progPathName);
000564                  WriteLn;
000565
000566                  REPEAT
000567                      blkType := scanner.ReadNumber(1);
000568                      blkSize := scanner.ReadNumber(3) - 4;
000569
000570                      CASE blkType OF
000571                          bSegTable:
```

```
000572                          BEGIN
000573                          nSegments := scanner.ReadNumber(2);
000574                          blkSize := blkSize-2;
000575                          FOR i := 1 TO nSegments DO
000576                              BEGIN
000577                              scanner.XferSequential(xRead, @segblk, SIZEOF(SegTblEntry));
000578                              blkSize := blkSize - scanner.actual;
000579                              XferLeft(Ptr(@segblk.segName), Ptr(ORD(@segName)+1), 8);
000580                              segNames.PutAt(segblk.SegNumber, @segName);
000581                              END;
000582                          END;
000583
000584                      bModuleName:
000585                          BEGIN
000586                          scanner.Skip(modNameSkip);
000587                          blkSize := blkSize-modNameSkip;
000588
000589                          scanner.XferSequential(xRead, Ptr(ORD(@segName)+1), 8);
000590                          blkSize := blkSize - scanner.actual;
000591                          IF segName = allBlanks THEN
000592                              segName := blankSeg;
000593                          END;
000594
000595                      bCodeBlock:
000596                          BEGIN
000597                          addr := scanner.ReadNumber(4);
000598                          blkSize := blkSize - 4;
000599                          segNames.PutAt(addr DIV $20000, @segName);
000600                          END;
000601                      END;
000602
000603                  scanner.Skip(blkSize);
000604              UNTIL scanner.atEnd  OR (blkType = bEOFMark);
000605
000606              scanner.Close;
000607              END;
000608      END;
000609
000610  (**********
000611      PROCEDURE ReadSegNames;
000612          LABEL
000613              1;
000614
000615          VAR prcsInfo:       ProcInfoRec;
000616              error:          INTEGER;
000617              progVolume:     TFilePath;
000618              fileName:       TFilePath;
000619              progExt:        TFilePath;
```

```
000620                  segNameFile:    TEXT;
000621                  i:              INTEGER;
000622                  segName:        S8;
000623                  segNum:         INTEGER;
000624          BEGIN
000625              Info_Process(error, My_id, prcsInfo);
000626              IF error <= 0 THEN
000627                  BEGIN
000628                  i := Length(prcsInfo.progPathName);
000629                  WHILE i > 0 DO
000630                      IF (prcsInfo.progPathName[i] = '}') OR (prcsInfo.progPathName[i] = '.') THEN
000631                          GOTO 1
000632                      ELSE
000633                          i := i - 1;
000634  1:
000635                  IF i > 0 THEN
000636                      filename := Concat(Copy(prcsInfo.progPathName, 1, i), 'SegNames.Text');
000637                  Reset(segNameFile, fileName);
000638                  i := IoResult;
000639                  IF i > 0 THEN
000640                      WriteLn('Unable to open ', fileName, ' because of error number ', i:1)
000641                  ELSE
000642                      BEGIN
000643                      WriteLn('Reading segment numbers and names from ', fileName);
000644                      WriteLn;
000645                      segNames := TArray.CREATE(NIL, mainHeap, 127, SIZEOF(S8));
000646                      segNames.InsNullsAt(1, 127);
000647
000648                      WHILE (i = 0) AND NOT Eof(segNameFile) DO
000649                          BEGIN
000650                          segNum := 0;
000651                          ReadLn(segNameFile, segNum, inStr);
000652                          i := IoResult;
000653                          IF (i <= 0) AND (1 <= segNum) AND (segNum <= 127) THEN
000654                              BEGIN
000655                              TrimBlanks(@inStr);
000656                              segName := Copy(Concat(inStr, '       '), 1, 8);
000657                              segNames.PutAt(segNum, @segName);
000658                              END
000659                          ELSE
000660                          IF i > 0 THEN
000661                              WriteLn('*** IoError number ', i:1, ' reading ', fileName)
000662                          ELSE
000663                          IF NOT Eof(segNameFile) THEN
000664                              WriteLn('*** Bad segment number: ', segNum:1, ' in file ', fileName);
000665                          END;
000666
000667                      segName := '????????';
```

```
000668                      FOR segNum := 1 TO 127 DO
000669                          IF TPByte(segNames.At(segNum))^ = 0 THEN
000670                              segNames.PutAt(segNum, @segName);
000671                      END;
000672
000673                  WriteLn;
000674                  WriteLn;
000675                  Close(segNameFile);
000676                  END;
000677          END;
000678  **********)
000679
000680          PROCEDURE WriteName;
000681          BEGIN
000682              IF mthName = '' THEN
000683                  BEGIN
000684                  LIntToHex(pc, @hexPC);
000685                  Write('$', hexPC, '        ');
000686                  END
000687              ELSE
000688              IF clsName = '' THEN
000689                  Write(mthName, '         ')
000690              ELSE
000691                  Write(clsName, '.', mthName);
000692          END;
000693
000694          FUNCTION TallyRange(pc1, pc2: LONGINT): INTEGER;
000695              VAR slot:   INTEGER;
000696          BEGIN   {If this proves too slow, make the sortarray by PC and binary search it}
000697              FOR slot := 1 TO tallies^^.header.size DO
000698                  WITH tallies^^.recs[slot] DO
000699                      IF count > 0 THEN
000700                          IF pc1 < epPC THEN
000701                              IF epPC <= pc2 THEN
000702                                  BEGIN
000703                                  TallyRange := slot;
000704                                  EXIT(TallyRange);
000705                                  END;
000706              TallyRange := 0;
000707          END;
000708
000709      PROCEDURE SwapIn(valueString: S8);
000710          BEGIN
000711          END;
000712
000713  BEGIN
000714      FOR j := tabLevel - 1 DOWNTO 0 DO (*^*)
000715          BEGIN
```

```
000716          elapsed := stopTime - traceTimes[j];
000717          FOR i := j - 1 DOWNTO 0 DO (*^*)
000718              traceTimes[i] := traceTimes[i] + elapsed;
000719          Tally(TpLONGINT(traceFrames[j] + 4)^, elapsed);
000720          END;
000721
000722      totalTime := stopTime - startTime(* - debugTime*);
000723      WriteLn;
000724      WriteLn((totalTime + 500) DIV 1000:1, ' milliseconds have elapsed since tallying began');
000725  (*  WriteLn((totalTime + 500) DIV 1000:1, ' milliseconds have elapsed since tallying began, not counting ',
000726              debugTime:1, ' ms. in debug code.');*)
000727
000728      WriteLn;
000729
000730      totalCalls := 0;
000731      callees := 0;
000732      FOR segNum := 1 TO 127 DO
000733          BEGIN
000734          segCount[segNum] := 0;
000735          segTime[segNum] := 0;
000736          END;
000737
000738      FOR slot := 1 TO tallies^^.header.size DO
000739          WITH tallies^^.recs[slot] DO
000740              IF count > 0 THEN
000741                  BEGIN
000742                  totalCalls := totalCalls + count;
000743                  callees := callees + 1;
000744                  segNum := TpINTEGER(@epPC)^ DIV 2;
000745                  segCount[segNum] := segCount[segNum] + count;
000746                  segTime[segNum] := segTime[segNum] + microseconds;
000747                  END;
000748
000749      IF totalCalls = 0 THEN
000750          WriteLn('All tallies are zero.')
000751      ELSE
000752          BEGIN {totalCalls > 0}
000753          roundOff := totalTime DIV 2;
000754
000755          WriteLn(callees:1, ' methods were called  a total of ', totalCalls:1, ' times.');
000756          WriteLn;
000757
000758          IF segNames = NIL THEN
000759              ReadSegNames;
000760
000761          WriteLn('                  SEGMENT USAGE');
000762          WriteLn;
000763
```

```
000764          WriteLn('No. of calls  % of time    Segment   SegSize   Seg#');
000765          WriteLn('------------  ---------    -------   -------   ----');
000766          WriteLn;
000767
000768          FOR segNum := 1 TO 127 DO
000769              IF segCount[segNum] > 0 THEN
000770                  BEGIN
000771                  IF segNames = NIL THEN
000772                      segName := '????????'
000773                  ELSE
000774                      segName := TPString(segNames.At(segNum))^;
000775
000776                      {Be sure the code is swapped in, before getting the size of the segment}
000777            {$R-} SwapIn(TpS8($20000 * segNum)^); {$IFC fRngObject} {$R+} {$ENDC}
000778
000779                  WriteLn(segCount[segNum]:8, ' ....... ',
000780                          (LONGINT(segTime[segNum]) * 100 + roundOff) DIV totalTime:3, '% ...  ',
000781                          segName,
000782                          LIntAndLint(TpLONGINT($20000 * segNum)^, $00FFFFFF):8,
000783                          segNum:7);
000784                  END;
000785
000786          REPEAT
000787              WriteLn;
000788              WriteLn;
000789
000790              Write('Report procedure executions sorted by (C = # Calls; T = % of Time; S = Segment)? ');
000791              ReadLn(inStr);
000792              StrUpperCased(@inStr);
000793              TrimBlanks(@inStr);
000794              IF inStr = '' THEN
000795                  sortBy := -1
000796              ELSE
000797              IF inStr[1] = 'C' THEN
000798                  sortBy := 1
000799              ELSE
000800              IF inStr[1] = 'T' THEN
000801                  sortBy := 2
000802              ELSE
000803              IF inStr[1] = 'S' THEN
000804                  sortBy := 3
000805              ELSE
000806                  sortBy := 0;
000807
000808              IF sortBy > 0 THEN
000809                  BEGIN {sortBy > 0}
000810                  sortKeys := MakeIdxArray(callees, FALSE);
000811
```

```
000812                  {$R-}
000813                  WITH sortKeys^^, tallies^^ DO
000814                      BEGIN {with}
000815                      i := 0;
000816                      FOR slot := 1 TO header.size DO
000817                          IF recs[slot].count > 0 THEN
000818                              BEGIN
000819                              i := i + 1;
000820                              records[i] := slot;
000821                              END;
000822
000823                      FOR i := 1 TO callees-1 DO
000824                          BEGIN
000825                          sloti := records[i];
000826                          FOR j := i+1 TO callees DO
000827                              BEGIN
000828                              slotj := records[j];
000829                              CASE sortBy OF
000830                                  1: swapem := recs[sloti].count > recs[slotj].count;
000831                                  2: swapem := recs[sloti].microseconds > recs[slotj].microseconds;
000832                                  3: swapem := TpINTEGER(@recs[sloti].epPC)^ DIV 2 >
000833                                               TpINTEGER(@recs[slotj].epPC)^ DIV 2;
000834                                  END;
000835                              IF swapem THEN
000836                                  BEGIN
000837                                  records[i] := slotj;
000838                                  records[j] := sloti;
000839                                  sloti := records[i];
000840                                  slotj := records[j];
000841                                  END;
000842                              END;
000843                          END;
000844                      END; {with}
000845                  {$IFC fRngObject}{$R+}{$ENDC}
000846
000847                  WriteLn('No. of calls  % of time      Routine name        PC      Segment');
000848                  WriteLn('------------  ---------   -----------------   --------   -------');
000849                  WriteLn;
000850
000851                  FOR i := callees DOWNTO 1 DO
000852                      BEGIN
000853                      slot := sortKeys^^.records[i];
000854                      WITH tallies^^.recs[slot] DO
000855                          BEGIN
000856                          calls := count;
000857                          micSecs := microseconds;
000858                          pc := epPC;
000859                          END;
```

```
000860
000861                  j := TpINTEGER(@pc)^ DIV 2;
000862                  IF sortBy = 3 THEN {if by segment}
000863                      IF i < callees THEN {if not first line printed}
000864                          IF j <> segNum THEN {if different from segment of previous line}
000865                              WriteLn; {then leave a blank line}
000866                  segNum := j;
000867
000868                  Write(calls:8, ' ..... ');
000869
000870                  pctg := (LONGINT(micSecs) * 100 + roundOff) DIV totalTime;
000871                  IF pctg = 0 THEN
000872                      Write('    ')
000873                  ELSE
000874                      Write(pctg:3, '%');
000875
000876                  Write(' ... ');
000877
000878                  IF GetDollarD(TppINTEGER(ORD(@pc)-4), clsName, mthName, nextPC) THEN;
000879                  WriteName;
000880
000881                  LIntToHex(pc, @hexPC);
000882                  Write('   ', hexPC);
000883
000884                  IF segNames = NIL THEN
000885                      Write(segNum:10)
000886                  ELSE
000887                      BEGIN
000888                      segName := TPString(segNames.At(segNum))^;
000889                      IF segName = '????????' THEN
000890                          Write(segNum:10)
000891                      ELSE
000892                          Write('   ', segName);
000893                      END;
000894
000895                  WriteLn;
000896
000897                  IF CheckKeyPress('Tally and Time Listing') THEN
000898                      i := 1;
000899                  END; {for i}
000900
000901              TArray(sortKeys).Free;
000902              END; {IF sortBy > 0}
000903          UNTIL sortBy < 0;
000904
000905      IF segNames <> NIL THEN {segNames will be non-NIL except in case of an IO error}
000906          REPEAT
000907              Write('List procedures that were and weren''t called in segment [name or number]? ');
```

```
000908                  ReadLn(inStr);
000909                  TrimBlanks(@inStr);
000910                  StrUpperCased(@inStr);
000911                  IF inStr = '?' THEN
000912                      BEGIN
000913                      WriteLn('List of all segments used by application:');
000914                      i := 0; {# output so far}
000915                      FOR segnum := 1 TO 127 DO
000916                          BEGIN
000917                          segname := TPString(segNames.At(segnum))^;
000918                          IF segname <> '' THEN
000919                              BEGIN
000920                              Write(segnum:3, ':', segname, '  ');
000921                              i := i+1;
000922                              IF i MOD 5 = 0 THEN
000923                                  WriteLn;
000924                              END;
000925                          END;
000926                      WriteLn;
000927                      END
000928                  ELSE
000929                  IF inStr <> '' THEN
000930                      BEGIN
000931                      StrToInt(@inStr, segNum, cState);
000932                      IF cState <> cvValid THEN
000933                          BEGIN
000934                          segNum := 0;
000935                          FOR i := 1 TO 127 DO
000936                              BEGIN
000937                              segName := TPString(segNames.At(i))^;
000938                              StrUpperCased(@segName);
000939                              IF segName = inStr THEN
000940                                  segNum := i;
000941                              END; {For i}
000942                          END  {invalid number}
000943                      ELSE IF (segNum>=1) AND (segNum<=127) THEN{make sure the segment number is OK}
000944                          BEGIN
000945                          segName := TPString(segNames.At(segNum))^;
000946                          IF segName = '' THEN
000947                              segNum := 0;
000948                          END
000949                      ELSE
000950                          segNum := 0;
000951
000952                      IF segNum = 0 THEN
000953                          WriteLn('No such segment')
000954                      ELSE
000955                          FOR wantCalled := TRUE DOWNTO FALSE DO
```

```
000956                          BEGIN
000957                          WriteLn;
000958                          Write('PROCEDURES THAT WERE ');
000959                          IF wantCalled THEN
000960                              Write('CALLED')
000961                          ELSE
000962                              Write('NOT CALLED OR HAD NO BP/EP');
000963                          Write(' IN SEGMENT');
000964
000965                          IF segNames <> NIL THEN
000966                              Write(': ', TPString(segNames.At(segNum))^);
000967                          WriteLn(' #', segNum:1, '--');
000968                          WriteLn;
000969
000970                          pc := segNum * $20000;
000971                          j := 0;
000972                          WHILE GetDollarD(TppINTEGER(ORD(@pc)-4), clsName, mthName, nextPC) DO
000973                              BEGIN
000974                              IF (TallyRange(pc, nextPC) > 0) = wantCalled THEN
000975                                  BEGIN
000976                                  WriteName;
000977                                  Write('   ');
000978                                  j := j + 1;
000979                                  IF j = 4 THEN
000980                                      BEGIN
000981                                      j := 0;
000982                                      WriteLn;
000983                                      END;
000984                                  END;
000985                              pc := nextPC;
000986                              IF CheckKeyPress('Segment Listing') THEN
000987                                  pc := 0;
000988                              END;
000989                          WriteLn;
000990                          WriteLn;
000991                          END; {FOR wantCalled}
000992                      END; {inStr <> ''}
000993
000994                  UNTIL inStr = '';
000995
000996          END; {totalCalls > 0}
000997
000998      WriteLn;
000999  END;
001000
001001  {$ENDC}
001002
001003  { ========================================================================================== }
```

```
001004
001005
001006  {$S SgCLAdbg} {for rest of file}
001007
001008
001009  { ================================== "FIELDS" METHODS ================================== }
001010
001011
001012  {$IFC fDebugMethods}
001013  PROCEDURE ParseDecl(inStr: S255;
001014                  PROCEDURE FoundName(token: S8);
001015                  PROCEDURE FoundType(token: S8; typeCode: TTypeCode; numBytes: INTEGER;
001016                   {for arrays only:} lowerBound, upperBound: INTEGER; memberTypeStr: S255);
001017                  PROCEDURE FoundUnexpected(token, wanted: S8));
001018      VAR p:          INTEGER;
001019          token:      S8;
001020          eoi:        INTEGER;
001021          alpha:      BOOLEAN;
001022          start:      INTEGER;    {where the last token started}
001023          inhibited:  BOOLEAN;
001024
001025      PROCEDURE NextToken;
001026      BEGIN
001027        {Skip leading blanks}
001028          WHILE (p <= eoi) AND (inStr[p] <= ' ') DO
001029              p := p + 1;
001030
001031          start := p;
001032
001033          IF p > eoi THEN
001034              token := ''
001035          ELSE
001036              BEGIN
001037              WHILE (p <= eoi) AND (inStr[p] IN ['-', '0'..'9', 'A'..'Z', 'a'..'z']) DO  {Form a word or number}
001038                  p := p + 1;
001039              alpha := p > start;
001040              IF NOT alpha THEN  {A single non-alphanumeric nonblank character}
001041                  p := p + 1;
001042              token := Copy(inStr, start, CMin(8, p - start));
001043              END;
001044      END;
001045
001046      PROCEDURE Expect(str: S8);
001047      BEGIN
001048          StrUpperCased(@token);
001049          IF token <> str THEN
001050              FoundUnexpected(token, str);
001051          NextToken;
```

```
001052      END;
001053
001054      FUNCTION ParseNumber: LONGINT;
001055          VAR k:      LONGINT;
001056              cState: TConvResult;
001057      BEGIN
001058          StrToLint(@token, k, cState);
001059          IF cState = cvValid THEN
001060              ParseNumber := k
001061          ELSE
001062              FoundUnexpected(token, 'a number');
001063          NextToken;
001064      END;
001065
001066      PROCEDURE ParseField; FORWARD;
001067
001068      PROCEDURE ParseType(inhibit: BOOLEAN);
001069          VAR typeName:       S8;
001070              upName:         S8;
001071              alphaName:      BOOLEAN;
001072              word:           S8;
001073              lowerBound:     INTEGER;
001074              upperBound:     INTEGER;
001075              pp:             INTEGER;
001076              i:              INTEGER;
001077              len:            INTEGER;
001078              wasInhibited:   BOOLEAN;
001079      BEGIN
001080          wasInhibited := inhibited;
001081          IF inhibit THEN
001082              inhibited := TRUE;
001083          typeName := token;
001084          upName := token;
001085          StrUpperCased(@upName);
001086          alphaName := alpha;
001087          NextToken;
001088          IF NOT alphaName THEN
001089              FoundUnexpected(typeName, 'typename')
001090          ELSE
001091          IF upName = 'RECORD' THEN
001092              BEGIN
001093              REPEAT
001094                  ParseField;
001095                  word := token;
001096                  StrUpperCased(@word);
001097                UNTIL (word = 'END') OR (word = '');
001098              Expect('END');
001099              END
```

```
001100          ELSE
001101          IF upName = 'ARRAY' THEN
001102              BEGIN
001103              Expect('[');
001104              lowerBound := ParseNumber;
001105              Expect('.');
001106              Expect('.');
001107              upperBound := ParseNumber;
001108              Expect(']');
001109              pp := p;
001110              Expect('OF');
001111              ParseType(TRUE);
001112              IF NOT inhibited THEN
001113                  FoundType('ARRAY', yArray, 0, lowerBound, upperBound, Copy(inStr, pp, start - pp));
001114              END
001115          ELSE
001116          IF upName = 'STRING' THEN
001117              BEGIN
001118              Expect('[');
001119              len := ParseNumber;
001120              Expect(']');
001121              IF NOT inhibited THEN
001122                  FoundType('STRING', yString, len + 1, 0, 0, '');
001123              END
001124          ELSE
001125          IF upName = 'BOOLEAN' THEN
001126              BEGIN
001127              IF NOT inhibited THEN
001128                  FoundType(typeName, yBoolean, 1, 0, 0, '');
001129              END
001130          ELSE
001131          IF upName = 'CHAR' THEN
001132              BEGIN
001133              IF NOT inhibited THEN
001134                  FoundType(typeName, yChar, 2, 0, 0, '')
001135              END
001136          ELSE
001137          IF upName = 'BYTE' THEN
001138              BEGIN
001139              IF NOT inhibited THEN
001140                  FoundType(typeName, yByte, 1, 0, 0, '')
001141              END
001142          ELSE
001143          IF upName = 'HEXBYTE' THEN
001144              BEGIN
001145              IF NOT inhibited THEN
001146                  FoundType(typeName, yHexByte, 1, 0, 0, '')
001147              END
```

```
001148          ELSE
001149          IF upName = 'INTEGER' THEN
001150              BEGIN
001151              IF NOT inhibited THEN
001152                  FoundType(typeName, yInteger, 2, 0, 0, '')
001153              END
001154          ELSE
001155          IF upName = 'HEXINTEG' THEN
001156              BEGIN
001157              IF NOT inhibited THEN
001158                  FoundType(typeName, yHexInteger, 1, 0, 0, '')
001159              END
001160          ELSE
001161          IF upName = 'LONGINT' THEN
001162              BEGIN
001163              IF NOT inhibited THEN
001164                  FoundType(typeName, yLongInt, 4, 0, 0, '')
001165              END
001166          ELSE
001167          IF upName = 'REAL' THEN
001168              BEGIN
001169              IF NOT inhibited THEN
001170                  FoundType(typeName, yReal, 4, 0, 0, '')
001171              END
001172          ELSE
001173          IF upName = 'POINT' THEN
001174              BEGIN
001175              IF NOT inhibited THEN
001176                  FoundType(typeName, yPoint, 4, 0, 0, '')
001177              END
001178          ELSE
001179          IF upName = 'PTR' THEN
001180              BEGIN
001181              IF NOT inhibited THEN
001182                  FoundType(typeName, yPtr, 4, 0, 0, '')
001183              END
001184          ELSE
001185          IF upName = 'LONGREAL' THEN
001186              BEGIN
001187              IF NOT inhibited THEN
001188                  FoundType(typeName, yLongReal, 8, 0, 0, '')
001189              END
001190          ELSE
001191          IF upName = 'LPOINT' THEN
001192              BEGIN
001193              IF NOT inhibited THEN
001194                  FoundType(typeName, yLPoint, 8, 0, 0, '')
001195              END
```

```
001196          ELSE
001197          IF upName = 'RECT' THEN
001198              BEGIN
001199              IF NOT inhibited THEN
001200                  FoundType(typeName, yRect, 8, 0, 0, '')
001201              END
001202          ELSE
001203          IF upName = 'LRECT' THEN
001204              BEGIN
001205              IF NOT inhibited THEN
001206                  FoundType(typeName, yLRect, 16, 0, 0, '')
001207              END
001208          ELSE
001209              BEGIN
001210              IF CiOfCn(upName) > 0 THEN
001211                  BEGIN
001212                  word := token;
001213                  StrUpperCased(@word);
001214                  IF word = 'OF' THEN
001215                      BEGIN
001216                      pp := p;
001217                      NextToken;
001218                      ParseType(TRUE);
001219                      IF NOT inhibited THEN
001220                          FoundType(typeName, yObject, SIZEOF(Handle), 0, 0, Copy(inStr, pp, start - pp));
001221                      END
001222                  ELSE
001223                  IF NOT inhibited THEN
001224                      FoundType(typeName, yObject, SIZEOF(Handle), 0, 0, '');
001225                  END
001226              ELSE
001227                  FoundUnexpected(typeName, 'typename');
001228              END;
001229          inhibited := wasInhibited;
001230      END;
001231
001232      PROCEDURE ParseField;
001233      BEGIN
001234          IF NOT alpha THEN
001235              BEGIN
001236              FoundUnexpected(token, 'var name');
001237              NextToken;
001238              END
001239          ELSE
001240              BEGIN
001241              IF NOT inhibited THEN
001242                  FoundName(token);
001243              NextToken;
```

```
001244              Expect(':');
001245              ParseType(FALSE);
001246              IF token = ';' THEN
001247                  NextToken
001248              ELSE
001249              IF (token <> '') AND (token <> 'END') THEN
001250                  FoundUnexpected(token, '; or END');
001251              END;
001252      END;
001253
001254  BEGIN
001255      inhibited := FALSE;
001256      p := 1;
001257      eoi := Length(inStr);
001258      Insert(' ', inStr, Length(inStr) + 1); {So that inStr[eoi+1] won't blow up}
001259      NextToken;
001260      WHILE token <> '' DO
001261          ParseField;
001262  END;
001263
001264
001265  PROCEDURE WriteDRecord{(numLevels: INTEGER; hDRecord: Handle; posInDRecord: INTEGER;
001266                          PROCEDURE SupplyFields(PROCEDURE Field(nameAndType: S255)))};
001267
001268      VAR fieldInDRecord: INTEGER;
001269
001270      PROCEDURE WrCkAbort;
001271      BEGIN
001272          IF KeyPress THEN
001273              BEGIN
001274              WrStr('...abort...');
001275              EXIT(WriteDRecord);
001276              END;
001277      END;
001278
001279      PROCEDURE DeclName(token: S8);
001280      BEGIN
001281          WrCkAbort;
001282          WrStr(Concat(token, ': '));
001283      END;
001284
001285      PROCEDURE SkipName(token: S8);
001286      BEGIN
001287          WrCkAbort;
001288      END;
001289
001290      PROCEDURE DeclBad(token, wanted: S8);
001291      BEGIN
```

```
001292          WrCkAbort;
001293          WrLn;
001294          WrStr('<<The Field proc expected:');
001295          WrStr(Concat(' ''', wanted, ''' '));
001296          WrStr('but encountered:');
001297          WrStr(Concat(' ''', token, '''>>'));
001298      END;
001299
001300      PROCEDURE DeclType(token: S8; typeCode: TTypeCode; numBytes: INTEGER;
001301                         lowerBound, upperBound: INTEGER; memberTypeStr: S255); FORWARD;
001302
001303      PROCEDURE DeclArray(token: S8; lowerBound, upperBound: INTEGER; memberTypeStr: S255);
001304          VAR str1:           S8;
001305              str2:           S8;
001306              i:              INTEGER;
001307              origPos:        INTEGER;
001308      BEGIN
001309          IF Odd(posInDRecord) THEN
001310              posInDRecord := posInDRecord + 1;
001311          IntToStr(lowerBound, @str1);
001312          IntToStr(upperBound, @str2);
001313          WrStr(Concat(token, ' [', str1, '..', str2, '] = {'));
001314          FOR i := lowerBound TO upperBound DO
001315              BEGIN
001316              IF i > lowerBound THEN
001317                  WrStr(', ');
001318              origPos := posInDRecord;
001319              IntToStr(i, @str1);
001320              ParseDecl(CONCAT(str1, ': ', memberTypeStr), DeclName, DeclType, DeclBad);
001321  (*****
001322              IF Odd(posInDRecord) THEN
001323                  posInDRecord := posInDRecord + 1;
001324  *****)
001325              END;
001326          WrStr('}');
001327      END;
001328
001329      PROCEDURE DeclType(token: S8; typeCode: TTypeCode; numBytes: INTEGER;
001330                         lowerBound, upperBound: INTEGER; memberTypeStr: S255);
001331          TYPE
001332              TAlias =
001333                  RECORD
001334                      CASE TTypeCode OF
001335                          yByte:      (asByte:    Byte);
001336                          yChar:      (asChar:    CHAR);
001337                          yInteger:   (asInteger: INTEGER);
001338                          yLongInt:   (asLongInt: LONGINT);
001339                          yLPoint:    (asLPoint:  FakeLPoint);
```

```
001340                      yLRect:     (asLRect:   FakeLRect);
001341                      yObject:    (asObject:  TObject);
001342                      yPoint:     (asPoint:   FakePoint);
001343                      yReal:      (asReal:    REAL);
001344                      yRect:      (asRect:    FakeRect);
001345                      yString:    (asString:  S255);
001346                      END;
001347          VAR alias:          ^TAlias;    {a bona fide use for aliasing instead of typecasting}
001348              obj:            TObject;
001349              str:            S255;
001350              i:              INTEGER;
001351      BEGIN
001352          IF typeCode = yArray THEN
001353              BEGIN
001354              DeclArray(token, lowerBound, upperBound, memberTypeStr);
001355              EXIT(DeclType);
001356              END;
001357
001358          IF token <> '' THEN
001359              WrStr(Concat(token, ' = '));
001360
001361          IF numBytes > 1 THEN
001362              IF Odd(posInDRecord) THEN
001363                  posInDRecord := posInDRecord + 1;
001364
001365          alias := POINTER(ORD(hDRecord^) + posInDRecord); {Careful, this is a relocatable location!}
001366          str := '';
001367          CASE typeCode OF
001368              yPtr:       BEGIN
001369                          LIntToHex(alias^.asLongInt, @str);
001370                          str := Concat('$', str);
001371                          END;
001372              yBoolean:   IF alias^.asByte = ORD(FALSE) THEN
001373                              str := 'FALSE'
001374                          ELSE
001375                              str := 'TRUE';
001376              yByte:      IntToStr(alias^.asByte, @str);
001377              yHexByte:   BEGIN
001378                          LIntToHex(alias^.asByte, @str);
001379                          str := CONCAT('$', Copy(str, 7, 2));
001380                          END;
001381              yChar:      BEGIN
001382                          str := 'A';
001383                          str[1] := alias^.asChar;
001384                          END;
001385              yInteger:   IntToStr(alias^.asInteger, @str);
001386              yHexInteger:BEGIN
001387                          LIntToHex(alias^.asInteger, @str);
```

```
001388                          str := CONCAT('$', Copy(str, 5, 4));
001389                          END;
001390              yLongInt:   LIntToStr(alias^.asLongInt, @str);
001391              yLPoint:    LPointToStr(alias^.asLPoint, @str);
001392              yLRect:     LRectToStr(alias^.asLRect, @str);
001393              yObject:    BEGIN
001394                          obj := alias^.asObject;
001395                          LIntToHex(ORD(obj), @str);
001396                          str := Concat('$', str, ' -- ');
001397                          IF obj = NIL THEN
001398                              str := 'NIL'
001399                          ELSE IF NOT ValidObject(Handle(obj)) THEN
001400                              str := Concat(str, 'Invalid Object')
001401                          ELSE
001402                              BEGIN
001403                              WrStr(str);
001404                              str := '';
001405                              obj.Debug(numLevels - 1, memberTypeStr);
001406                              END;
001407                          END;
001408              yPoint:     PointToStr(alias^.asPoint, @str);
001409              yRect:      RectToStr(alias^.asRect, @str);
001410              yString:    BEGIN
001411                          str := Concat('''', alias^.asString, '''');
001412                          IF Odd(numBytes) THEN
001413                              numBytes := numBytes + 1;
001414                          END;
001415              yReal:      BEGIN
001416                          LIntToStr(Round(alias^.asReal * 1000.0), @str);
001417                          FOR i := LENGTH(str) TO 3 DO
001418                              Insert('0', str, 1);
001419                          Insert('.', str, LENGTH(str)-2);
001420                          END;
001421              OTHERWISE   DeclBad(token, 'typename');
001422              END;
001423
001424          WrCkAbort;
001425          IF str <> '' THEN
001426              WrStr(Concat(str, ' '));
001427
001428          posInDRecord := posInDRecord + numBytes;
001429      END;
001430
001431      PROCEDURE DebugField(nameAndType: S255);
001432      BEGIN
001433          IF nameAndType <> '' THEN
001434              BEGIN
001435              fieldInDRecord := fieldInDRecord + 1;
```

```
001436              IF fieldInDRecord > 1 THEN
001437                  WrStr('; ');
001438              ParseDecl(nameAndType, DeclName, DeclType, DeclBad);
001439              WrCkAbort;
001440              END
001441          ELSE    {Empty string signifies padding to a word boundary, if necessary}
001442          IF Odd(posInDRecord) THEN
001443              posInDRecord := posInDRecord + 1;
001444      END;
001445
001446  BEGIN
001447      IF KeyPress THEN
001448          Exit(WriteDRecord);
001449
001450      fieldInDRecord := 0;
001451      WrStr('[ ');
001452      SupplyFields(DebugField);
001453      WrStr('] ');
001454  END;
001455
001456
001457  PROCEDURE DumpVar{(pVariable: Ptr; nameAndType: S255)};
001458
001459      PROCEDURE SupplyVar(PROCEDURE Field(nameAndType: S255));
001460      BEGIN
001461          Field(nameAndType);
001462      END;
001463
001464  BEGIN
001465      currXPos := 0;
001466      outputIndent := 20;
001467      WriteDRecord(1, @pVariable, 0, SupplyVar);
001468      outputIndent := 0;
001469      WrLn;
001470  END;
001471  {$ENDC}
001472
001473
001474  { ================================== KITBUG ==================================== }
001475
001476
001477  {$IFC fDbgObject}
001478  PROCEDURE WrStr{(str: S255)};   { Write a STRING with word-wrap }
001479      VAR start:  INTEGER;
001480          maxLen: INTEGER;
001481          len:    INTEGER;
001482          total:  INTEGER;
001483  BEGIN
```

```
001484      total := Length(str);
001485      start := 1;
001486      WHILE start <= total DO
001487          BEGIN
001488          len := total - start + 1;
001489          maxLen := outputRMargin - currXPos;
001490          IF len > maxLen THEN
001491              BEGIN
001492              len := maxLen;
001493              WHILE (len > 0) AND (str[len] <> ' ') DO
001494                  len := len - 1;
001495              IF (len = 0) AND (currXPos = outputIndent) THEN
001496                  len := maxLen;
001497              END;
001498          IF len > 0 THEN
001499              BEGIN
001500              Write(Copy(str, start, len));
001501              currXPos := currXPos + len;
001502              start := start + len;
001503              END;
001504          IF (currXPos >= outputRMargin) OR (start <= total) THEN
001505              WrLn;
001506          END;
001507  END;
001508
001509
001510  PROCEDURE WrLn;                     { goto next line and output indentation }
001511  BEGIN
001512      WriteLn;
001513      IF outputIndent > 0 THEN
001514          BEGIN
001515          Write(' ':outputIndent);
001516          currXPos := outputIndent;
001517          END
001518      ELSE
001519          currXPos := 0;
001520  END;
001521
001522
001523  FUNCTION CheckKeyPress{(routine: S255): BOOLEAN};
001524      VAR ch: CHAR;
001525  BEGIN
001526      IF KeyPress THEN
001527          BEGIN
001528          IF routine <> '' THEN
001529              BEGIN
001530              WriteLn;
001531              WriteLn('   -- ', routine, ' stopped because you typed a key --');
```

```
001532              WriteLn;
001533              END;
001534  (* commented out and should be removed if paslib bug has been fixed
001535          { flush characters; because of PASLIB bug, also stop when user types a ~ }
001536          ch := ' ';
001537          WHILE KeyPress AND (ch<>'~') DO
001538              IF EOLn THEN
001539                  ReadLn
001540              ELSE
001541                  Read(ch);
001542  *)
001543          CheckKeyPress := TRUE;
001544          END
001545      ELSE
001546          CheckKeyPress := FALSE;
001547  END;
001548
001549
001550      {$IFC fDebugMethods}
001551  PROCEDURE WrObj(object: TObject; numLevels: INTEGER; memberTypeStr: S255);
001552  BEGIN
001553      WriteLn;
001554      currXPos := 0;
001555      outputIndent := 0;
001556      IF ValidObject(Handle(object)) THEN
001557          BEGIN
001558          object.Debug(numLevels, memberTypeStr);
001559          IF CheckKeyPress('Display of the object') THEN;
001560          END
001561      ELSE
001562          Write('Not an object: ', ORD(object):1);
001563  END;
001564      {$ENDC}
001565
001566
001567  {$S SgCLAini}
001568  PROCEDURE DumpHeap(heap: THeap; wantedSTP: LONGINT; wantedReference: LONGINT; fPrintSelf: BOOLEAN);
001569      VAR hz:             THz;
001570          cb:             TC;
001571          hndl:           Handle;
001572          obj:            TObject;
001573          heapSize:       LONGINT;
001574          numObjects:     LONGINT;    {Clascal objects only}
001575          objOvhdSize:    LONGINT;    {includes master, header, and class pointer}
001576          objDataSize:    LONGINT;
001577          numOther:       LONGINT;    {Non-Clascal objects}
001578          otherSize:      LONGINT;
001579          numFree:        LONGINT;
```

```
001580          freeSize:       LONGINT;
001581          bigFreeSize:    LONGINT;
001582          bk:             TBk;
001583          dumpIt:         BOOLEAN;
001584          valid:          BOOLEAN;
001585          offset:         INTEGER;
001586          base:           LONGINT;
001587          hStr:           S8;
001588          class:          TClass;
001589          className:      TClassName;
001590  BEGIN
001591      WriteLn;
001592
001593      IF heap = NIL THEN
001594          BEGIN
001595          WriteLn('The heap pointer is NIL');
001596          WriteLn;
001597          EXIT(DumpHeap);
001598          END;
001599
001600      hz := THz(heap);
001601      heapSize := cbOfHz(hz);
001602
001603      numObjects := 0;
001604      objOvhdSize := 0;
001605      objDataSize := 0;
001606      numOther := 0;
001607      otherSize := 0;
001608      numFree := 0;
001609      freeSize := 0;
001610      bigFreeSize := 0;
001611
001612      WriteLn('Heap size in bytes: ', heapSize:6);
001613      WriteLn('Bytes free:         ', hz^.cbFree:6);
001614      WriteLn;
001615
001616      WriteLn('Heap contents (handle, size in bytes):');
001617      WriteLn;
001618
001619          { setup indentation for writing objects }
001620      outputIndent := 17; { '$', ORD(hndl):8, cb:6, ': ' }
001621
001622      bk := hz^.bkFst;
001623      WHILE (ORD(bk) >= ORD(hz^.bkFst)) AND (ORD(bk) <= ORD(hz^.bkLst)) DO
001624          BEGIN
001625          IF bk^.hdr.tybk <> tybkFree THEN
001626              cb := bk^.hdr.cw * 2
001627          ELSE
```

```
001628              cb := bk^.cwFree * 2;
001629
001630          IF cb <= 0 THEN
001631              BEGIN
001632              WriteLn('FREE BLOCK ', ORD(bk):1, ' HAS LENGTH', cb);
001633              EXIT(DumpHeap);
001634              END;
001635
001636          CASE bk^.hdr.tybk OF
001637
001638              tybkStd:
001639                  BEGIN
001640  {$IFC LibraryVersion <= 20}
001641                  hndl := Handle(ORD(hz) + bk^.oh);
001642  {$ELSEC}
001643                  hndl := Handle(ORD(@hz^.argpPool) + (LONGINT(bk^.bp.ip)*4));
001644  {$ENDC}
001645                  valid := ValidObject(hndl);
001646
001647                  IF wantedSTP > 0 THEN
001648                      IF valid THEN    {looks like a class pointer; pray that it is!}
001649                          dumpIt := %_InObCp(ORD(hndl), wantedSTP)
001650                      ELSE
001651                          dumpIt := FALSE
001652                  ELSE
001653                  IF wantedReference <> 0 THEN
001654                      BEGIN
001655                      offset := 0;
001656                      base := ORD(hndl^);
001657                      WHILE (offset < cb) AND (TpLONGINT(base + offset)^ <> wantedReference) DO
001658                          offset := offset + 2;
001659                      dumpIt := offset < cb;
001660                      END
001661                  ELSE
001662                      dumpIt := TRUE;
001663
001664                  IF dumpIt THEN
001665                      BEGIN
001666                      LIntToHex(ORD(hndl), @hStr);
001667                      Write('$', hStr, cb:6, ': ');
001668                      IF bk <> TBk(ORD(hndl^) - 4) THEN
001669                          BEGIN
001670                          WriteLn('INCORRECT BACK POINTER FOR bk = ', ORD(bk):1);
001671                          EXIT(DumpHeap);
001672                          END;
001673
001674                      IF valid THEN
001675                          BEGIN
```

```
001676                          obj := TObject(hndl);
001677
001678                          currXPos := outputIndent;
001679
001680                          {$IFC fDebugMethods}
001681                          IF fPrintSelf THEN
001682                              obj.Debug(1, '')
001683                          ELSE
001684                              obj.Debug(0, '');
001685                          {$ELSEC}
001686                          class := obj.Class;
001687                          CpToCn(TPSliceTable(class), TS8(className));
001688                          TrimBlanks(@className);
001689                          Write(className);
001690                          {$ENDC}
001691
001692                          numObjects := numObjects + 1;
001693                          objOvhdSize := objOvhdSize + 12; {master=4, header=4, classPtr=4}
001694                          objDataSize := objDataSize + cb-4;   {classPtr=4}
001695                          END
001696                      ELSE
001697                          BEGIN
001698                          numOther := numOther + 1;
001699                          otherSize := otherSize + cb;
001700                          Write('???');
001701                          END;
001702
001703                      WriteLn;
001704                      END;
001705                  bk := TBk(ORD(hndl^) - 4); {in case the heap compacted during obj.Debug}
001706                  END;
001707
001708              tybkFree:
001709                  BEGIN
001710                  numFree := numFree + 1;
001711                  freeSize := freeSize + cb;
001712                  IF cb > bigFreeSize THEN
001713                      bigFreeSize := cb;
001714                  END;
001715
001716              OTHERWISE
001717                  BEGIN
001718                  numOther := numOther + 1;
001719                  otherSize := otherSize + cb;
001720                  END;
001721              END;
001722
001723          bk := TBk(ORD(bk) + cb);
```

```
001724
001725          IF CheckKeyPress('HeapDump') THEN
001726              EXIT(DumpHeap);
001727          END;
001728      WriteLn;
001729
001730      IF numObjects > 0 THEN
001731          BEGIN
001732          WriteLn('Number of Clascal objects:         ', numObjects:6);
001733          IF wantedReference = 0 THEN
001734              BEGIN
001735              WriteLn('Bytes in their headers & masters:  ', objOvhdSize:12);
001736              WriteLn('Bytes in their records:            ', objDataSize:12);
001737              IF objDataSize+objOvhdSize > 0 THEN
001738                  WriteLn('Header and master overhead:        ',
001739                          (100 * objOvhdSize) DIV (objDataSize+objOvhdSize):5, '%');
001740              END;
001741          WriteLn;
001742          END;
001743
001744      IF (wantedSTP <= 0) AND (wantedReference = 0) THEN
001745         BEGIN
001746         WriteLn('Number of free blocks:             ', numFree:6);
001747         WriteLn('Largest free block:                ', bigFreeSize:6);
001748         WriteLn('Bytes in free blocks:              ', freeSize:12);
001749         WriteLn;
001750         WriteLn('Number of other blocks:            ', numOther:6);
001751         WriteLn('Bytes in those blocks:             ', otherSize:12);
001752         WriteLn;
001753         WriteLn('Other overhead:                    ', heapSize-objOvhdSize-objDataSize-freeSize-otherSize:12);
001754         WriteLn('Total heap size in bytes:          ', heapSize:12);
001755         WriteLn;
001756         END;
001757  END;
001758
001759
001760  {$S SgCLAini}
001761  PROCEDURE GoKitBug; {intended to be called from LisaBug}
001762  BEGIN
001763      EntDebugger(' ', 'Called from GoKitBug');
001764  END;
001765
001766
001767  {$S SgCLAini}
001768  PROCEDURE EntDebugger{(inputStr, enterReason: S255)};
001769      LABEL 99;
001770      CONST   null = CHR(0);
001771      VAR token:      S255;
```

```
001772          cState:     TConvResult;
001773          timeToGo:   BOOLEAN;
001774          brClass:    S8;
001775          brMethod:   S8;
001776
001777      PROCEDURE GetToken;
001778          VAR endOfToken: INTEGER;
001779      BEGIN
001780          token := '';
001781          WHILE Pos(' ', inputStr) = 1 DO
001782              Delete(inputStr,1,1);
001783          endOfToken := Pos(' ', inputStr)-1;
001784          IF endOfToken <= 0 THEN
001785              endOfToken := Length(inputStr);
001786          token := Copy(inputStr, 1, endOfToken);
001787          Delete(inputStr, 1, endOfToken);
001788      END;
001789
001790      PROCEDURE DebugStatus;
001791          VAR i: INTEGER;
001792      BEGIN
001793          IntToStr(curTraceLevel, @token);
001794          Write('Watch Level = ',token);
001795          IntToStr(defTraceCount, @token);
001796          WriteLn(', Watch Count = ',token);
001797          FOR i := 1 TO breakMCount DO
001798              WITH breakMethods[i] DO
001799                  IF (brClass <> '') OR (brMethod <> '') THEN
001800                      WriteLn(i:3, ': ', brClass:8,'.',brMethod:8)
001801      END;
001802
001803      PROCEDURE ClearBreaks;
001804          VAR brNumber:   INTEGER;
001805              cState:     TConvResult;
001806      BEGIN
001807          GetToken;
001808          IF token = '' THEN
001809              BEGIN
001810              Write('Clear which breakpoint [A for all breakpoints]? ');
001811              ReadLn(token);
001812              END;
001813
001814          TrimBlanks(@token);
001815          StrUpperCased(@token);
001816
001817          IF token <> '' THEN
001818              IF token[1] = 'A' THEN
001819                  breakMCount := 0
```

```
001820              ELSE
001821                  BEGIN
001822                  StrToInt(@token, brNumber, cState);
001823                  IF cState = cvValid THEN
001824                      IF (brNumber >= 1) AND (brNumber <= breakMCount) THEN
001825                          WITH breakMethods[brNumber] DO
001826                              BEGIN
001827                              brClass := '';
001828                              brMethod := '';
001829
001830                              IF brNumber = breakMCount THEN
001831                                  breakMCount := breakMCount - 1;
001832                              END;
001833                  END;
001834      END;
001835
001836      { get input up to the first '.'; if non-null convert to 8 characters;
001837          prompt user with argument if there is no pending input }
001838      PROCEDURE GetOne(prompt: S255);
001839          VAR i:  INTEGER;
001840      BEGIN
001841          GetToken;
001842
001843          IF token = '' THEN
001844              BEGIN
001845              Write(prompt);
001846              ReadLn(token);
001847              END;
001848
001849          i := Pos('.', token);
001850
001851          IF i > 0 THEN
001852              BEGIN
001853              inputStr := Concat(Copy(token, i + 1, Length(token) - i), inputStr);
001854              token := Copy(token, 1, i - 1);
001855              END;
001856
001857          TrimBlanks(@token);
001858          IF token <> '' THEN
001859              BEGIN
001860              StrUpperCased(@token);
001861              token := Copy(Concat(token, '       '), 1, 8);
001862              END;
001863      END;
001864
001865      PROCEDURE BrSetup(prompt: S255);
001866          VAR brNumber:   INTEGER;
001867
```

```
001868          FUNCTION MoreThanAClass: Boolean;
001869          BEGIN
001870              MoreThanAClass := TRUE;
001871              IF length(inputstr) > 0 THEN
001872                  IF inputstr[length(inputstr)] = '.' THEN
001873                  BEGIN
001874                      GetOne(Concat(prompt,' what Class?'));
001875                      WITH breakMethods[brNumber] DO
001876                          BEGIN
001877                          brClass := token;
001878                          brMethod := '';
001879                          END;
001880                      MoreThanAClass := FALSE;
001881                  END;
001882          END;
001883
001884      BEGIN
001885          FOR brNumber := 1 TO maxBreaks DO
001886              WITH breakMethods[brNumber] DO
001887                  IF (brNumber > breakMCount) OR ((brClass='') AND (brMethod='')) THEN
001888                      BEGIN
001889                      IF MoreThanAClass THEN
001890                          BEGIN
001891                          GetOne(Concat(prompt,' what Class?'));
001892                          brClass := token;
001893                          GetOne(Concat(prompt,' what Method?'));
001894                          brMethod := token;
001895                          END;
001896                      IF (brClass <> '') OR (brMethod <> '') THEN
001897                          breakMCount := Max(breakMCount, brNumber);
001898                      lastBpPc := 0;
001899                      lastEpPc := 0;
001900                      EXIT(BrSetup);
001901                      END;
001902
001903          WriteLn('Too Many Breaks Defined, you must first clear a breakpoint')
001904      END;
001905
001906      PROCEDURE TraceOrNot;
001907          VAR i: INTEGER;
001908      BEGIN
001909          GetToken;
001910          StrToInt(@token, i, cState);
001911          IF cState = cvValid THEN
001912              BEGIN
001913              defTraceCount := i;
001914              GetToken;
001915              END
```

```
001916          ELSE
001917              defTraceCount := 0;
001918
001919          returnToMain := TRUE;
001920          fTraceSelf := FALSE;
001921          fTraceClass := FALSE;
001922          WHILE token <> '' DO
001923              BEGIN
001924              StrUpperCased(@token);
001925              IF token[1] = 'A' THEN {Stay on Alternate Screen During Trace}
001926                  returnToMain := FALSE
001927              ELSE
001928              IF token[1] = 'C' THEN {Print Class with Trace}
001929                  fTraceClass := TRUE
001930              ELSE
001931              IF token[1] = 'F' THEN {Print Fields with Trace}
001932                  fTraceSelf := TRUE;
001933              GetToken;
001934              END;
001935
001936          fTraceEnabled := TRUE;
001937          traceCount := defTraceCount;
001938      END;
001939
001940      PROCEDURE Level;
001941          VAR i: INTEGER;
001942      BEGIN
001943          GetToken;
001944          IF token = '' THEN
001945              BEGIN
001946              Write('Lowest BP level to watch (1..9999)? ');
001947              ReadLn(token);
001948              END;
001949          StrToInt(@token, i, cState);
001950          IF cState = cvValid THEN
001951              IF (i >= 1) AND (i <= 32000) THEN
001952                  curTraceLevel := i;
001953      END;
001954
001955      FUNCTION  YesNo(prompt: S255): BOOLEAN;
001956      BEGIN
001957          REPEAT
001958              GetOne(Concat(prompt, '? [Y/N]: '));
001959              IF token = '' THEN
001960                  GOTO 99;
001961            UNTIL token[1] IN ['Y', 'N'];
001962          YesNo := token[1] = 'Y';
001963      END;
```

```
001964
001965      PROCEDURE PromptOnOff;
001966      BEGIN
001967          showPrompt := YesNo('Show Debugger Prompt');
001968      END;
001969
001970      {$IFC fDebugMethods}
001971      PROCEDURE Inspect;
001972      VAR lh: LONGINT;
001973          d:  INTEGER;
001974      BEGIN
001975          GetToken;
001976          IF token = '' THEN
001977              BEGIN
001978              Write('Handle to inspect [depth] [member decl]? ');
001979              Readln(inputStr);
001980              GetToken;
001981              END;
001982
001983          HexStrToLInt(@token, lh, cState);
001984          IF cState <> cvValid THEN
001985              BEGIN
001986              WriteLn('Not a hex number');
001987              Exit(Inspect);
001988              END
001989          ELSE
001990              GetToken;
001991
001992          StrToInt(@token, d, cState);
001993          IF cState <> cvValid THEN
001994              d := 1;
001995
001996          IF ValidObject(Handle(lh)) THEN
001997              BEGIN
001998              WrObj(TObject(lh), d, inputStr);
001999              Writeln;
002000              END
002001          ELSE
002002              Writeln('Invalid Object');
002003          Writeln;
002004      END;
002005      {$ENDC}
002006
002007
002008      PROCEDURE TallyAndTime;
002009      BEGIN
002010          tallyingCalls := FALSE;
002011          IF tallies <> NIL THEN
```

```
002012              BEGIN
002013              IF YesNo('Do you want to see performance measurements now') THEN
002014                  TallyReport;
002015              IF YesNo('Do you want to zero the tallies and times') THEN
002016                  TallyZero;
002017              END;
002018          IF YesNo('Do you want to continue execution and measure its performance') THEN
002019              TallyStart;
002020          WriteLn;
002021      END;
002022
002023      PROCEDURE RefsToObject;
002024      VAR lh: LONGINT;
002025      BEGIN
002026          GetToken;
002027          IF token = '' THEN
002028              BEGIN
002029              Write('Handle of the object whose every Reference from the same heap should be dumped? ');
002030              Readln(inputStr);
002031              GetToken;
002032              END;
002033
002034          HexStrToLInt(@token, lh, cState);
002035          IF cState <> cvValid THEN
002036              BEGIN
002037              WriteLn('Not a hex number');
002038              Exit(RefsToObject);
002039              END;
002040
002041          IF ValidObject(Handle(lh)) THEN
002042              DumpHeap(TObject(lh).Heap, -1, lh, TRUE)
002043          ELSE
002044              Writeln('Invalid Object');
002045          Writeln;
002046      END;
002047
002048      FUNCTION  StackFrame(whichFrame: INTEGER): LONGINT;
002049          { Returns address of stack frame 'whichFrame' (>=-1);
002050              whichFrame < 0 returns -1.
002051              whichFrame = 1 is the top frame not belonging to the debugger itself.
002052                  When called from ABCBREAK, the caller of ABCBREAK is frame 1;
002053                  When called from BEPSN, the caller of BP or EP is frame 1.
002054              whichFrame = 2 is the caller of frame 1, and so on.
002055              If whichFrame is greater than # frames, returns -1.
002056              If neither ABCBREAK nor BEPSN is on the stack, returns -1.}
002057          VAR dummy:          INTEGER;    { must be first local and two bytes long }
002058              RA6:            LONGINT;
002059              RA5:            LONGINT;
```

```
002060              i:              INTEGER;
002061              className:      TClassName;
002062              procName:       S8;
002063              startCount:     BOOLEAN;
002064              frameReference: INTEGER;
002065              nextPC:         LONGINT;
002066      BEGIN
002067          StackFrame := -1;       { default return }
002068          frameReference := 0;
002069          startCount := FALSE;
002070
002071          RA5 := %_GetA5;
002072          RA6 := ORD(@dummy)+2;   { stack frame called by current one; start with my stack frame }
002073          WHILE (whichFrame >= frameReference) AND (RA6 <> RA5) DO
002074              BEGIN
002075              IF NOT startCount THEN
002076                  BEGIN
002077                  IF GetDollarD(TppINTEGER(RA6), className, procName, nextPC) THEN { is this frame 0? }
002078                      IF (className = '') AND ((procName = 'BEPSN   ') OR (procName = 'ABCBREAK'))THEN
002079                          BEGIN
002080                          startCount := TRUE; { yes }
002081                          IF procName = 'BEPSN   ' THEN
002082                              frameReference := -1;
002083                          END;
002084                  END;
002085
002086              RA6 := TpLONGINT(RA6)^;    { preceding stack frame }
002087
002088              IF startCount THEN
002089                  BEGIN
002090                  IF whichFrame = frameReference THEN
002091                      StackFrame := RA6;
002092                  whichFrame := whichFrame - 1;
002093                  END;
002094              END;
002095      END;
002096
002097      PROCEDURE WrMemory(start: LONGINT; numBytes: INTEGER; checkAddresses: BOOLEAN);
002098          VAR addr:       LONGINT;
002099              str:        S255;
002100              asChars:    STRING[16];
002101              extdWord:   LONGINT;
002102              overflow:   BOOLEAN;
002103              fullBytes:  INTEGER;
002104
002105          FUNCTION Byte2Char(n: INTEGER): CHAR;
002106          BEGIN
002107              IF (n < 32) OR (n > ORD('~')) THEN
```

```
002108                Byte2Char := '•'
002109            ELSE
002110                Byte2Char := CHR(n);
002111        END;
002112
002113        {$R-}
002114        PROCEDURE AddCh(s: TPString; ch: CHAR; maxStrLeng: INTEGER; VAR overflow: BOOLEAN);
002115            BEGIN
002116            overflow := TRUE;
002117            IF Length(s^) < maxStrLeng THEN
002118                BEGIN
002119                overflow := FALSE;
002120                s^[0] := CHR(ORD(s^[0]) + 1);
002121                s^[ORD(s^[0])] := ch;
002122              END;
002123          END;
002124          {$IFC fRngObject} {$R+} {$ENDC}
002125
002126    BEGIN
002127        { start at an even address and fullBytes a multiple of 16 >= numBytes }
002128
002129        addr := (start DIV 2) * 2;
002130
002131        IF checkAddresses THEN
002132            IF NOT ValidDataAddress(addr) THEN
002133                IF NOT ValidGlobalAddress(addr) THEN
002134                    BEGIN
002135                    WriteLn;
002136                    Write('*** That address is neither in a data segment nor in the stack/global segment. ');
002137                    WriteLn('***');
002138                    EXIT(WrMemory);
002139                    END;
002140
002141        fullBytes := ((numBytes + 15) DIV 16) * 16;
002142
002143        WHILE fullBytes > 0 DO
002144            BEGIN
002145            IF fullBytes MOD 16 = 0 THEN
002146                BEGIN
002147                LIntToHex(addr, @str);
002148                Write('  ', str, '   ');
002149                asChars := '';
002150                END;
002151
002152            IF checkAddresses THEN
002153                IF NOT ValidDataAddress(addr) THEN
002154                    IF NOT ValidGlobalAddress(addr) THEN
002155                        WHILE numBytes > 0 DO
```

```
002156                          BEGIN
002157                          Write('•••• ');
002158                          AddCh(@asChars, '•', 16, overflow);
002159                          AddCh(@asChars, '•', 16, overflow);
002160                          numBytes := numBytes - 2;
002161                          fullBytes := fullBytes - 2;
002162                          END;
002163
002164              IF numBytes <= 0 THEN
002165                  WHILE fullBytes > 0 DO
002166                      BEGIN
002167                      Write('     ');
002168                      AddCh(@asChars, ' ', 16, overflow);
002169                      AddCh(@asChars, ' ', 16, overflow);
002170                      fullBytes := fullBytes - 2;
002171                      END;
002172
002173              IF fullBytes > 0 THEN
002174                  BEGIN
002175                  extdWord := LIntAndLInt(TpINTEGER(addr)^, $0000FFFF);
002176                  LIntToHex(extdWord, @str);
002177                  Delete(str, 1, 4);  {4 leading zeros}
002178                  Write(str, ' ');
002179
002180                  AddCh(@asChars, Byte2Char(extdWord DIV 256), 16, overflow);
002181                  AddCh(@asChars, Byte2Char(extdWord MOD 256), 16, overflow);
002182
002183                  addr := addr + 2;
002184                  fullBytes := fullBytes - 2;
002185                  numBytes := numBytes - 2;
002186                  END;
002187
002188              IF fullBytes MOD 16 = 0 THEN
002189                  WriteLn('   |', asChars, '|');
002190              END;
002191
002192          WriteLn;
002193      END;
002194
002195      FUNCTION WrFrame(whichFrame: INTEGER; full: BOOLEAN): BOOLEAN;
002196          { Write a frame given its number; return TRUE if that frame exists }
002197          VAR RA5:            LONGINT;
002198              calledA6:       LONGINT;
002199              addr:           LONGINT;
002200              laterA6:        LONGINT;
002201              earlierA6:      LONGINT;
002202              hexStr:         S8;
002203              gotMainProg:    BOOLEAN;
```

```
002204              className:      TClassName;
002205              methName:       S8;
002206              procName:       S255;
002207              procStart:      LONGINT;
002208              frameSELF:      TObject;
002209              class:          TClass;
002210              nextPC:         LONGINT;
002211              localBytes:     INTEGER;
002212              paramBytes:     INTEGER;
002213              selfBytes:      INTEGER;
002214
002215      PROCEDURE SwapIn(valueString: S8);
002216          BEGIN
002217          END;
002218
002219      BEGIN
002220          RA5 := %_GetA5;
002221          calledA6 := StackFrame(whichFrame-1);       { A6 of frame called by desired frame }
002222          IF (whichFrame < 1) OR (calledA6 = -1) OR (calledA6 = RA5) THEN
002223              BEGIN
002224              WrFrame := FALSE;
002225              EXIT(WrFrame);
002226              END;
002227          WrFrame := TRUE;
002228
002229          addr := calledA6;
002230          LIntToHex(TpLONGINT(addr)^, @hexStr);
002231          Write('Frame # ', whichFrame:3, ' @ $', hexStr, '  ');
002232          gotMainProg := TpLONGINT(addr)^ = RA5;    { stack frame for main prog starts at A5 }
002233
002234          { find called-from address }
002235          IF GetDollarD(TppINTEGER(calledA6), className, methName, nextPC) THEN
002236              IF className = '' THEN
002237                  procName := methName
002238              ELSE
002239                  procName := Concat(className, '.', methName)
002240          ELSE
002241              procName := '';
002242
002243          IF procName <> '' THEN
002244              BEGIN
002245              Write(procName:17);
002246
002247              { search back in code for TST.W <n>(A7) and LINK A6,<m> instructions }
002248              addr := calledA6+4;
002249              addr := TpLONGINT(addr)^;
002250        {$R-} SwapIn(TPS8(addr)^); {$IFC fRngObject} {$R+} {$ENDC} {Be sure the code is swapped in}
002251              procStart := 0;
```

```
002252              WHILE procStart = 0 DO
002253                  BEGIN
002254                  addr := addr - 2;
002255
002256                  IF TpINTEGER(addr)^ = $4E56 { LINK A6,<n> } THEN
002257                      { found LINK, so numLocal is now set correctly, and
002258                        start of PROCEDURE is 4 bytes back (auto stack expansion) }
002259                      procStart := addr - 4;
002260                  END;
002261
002262              IF gotMainProg THEN
002263                  procStart := procStart + 4;         { main prog has no stack expansion }
002264
002265              addr := calledA6+4;
002266              LintToHex(TpLONGINT(addr)^-4 - procStart, @hexStr);
002267              Delete(hexStr, 1, Length(hexStr)-4);       { only want the lower 4 digits of hex number }
002268              Write('+ $', hexStr);
002269
002270              { advance to next stack frame now, so we can get at its variables }
002271              laterA6 := calledA6;
002272              calledA6 := TpLONGINT(laterA6)^;
002273              IF calledA6 = RA5 THEN
002274                  earlierA6 := RA5
002275              ELSE
002276                  earlierA6 := TpLONGINT(calledA6)^;
002277
002278              frameSELF := NIL;
002279              IF (className <> '') AND (procName <> 'CREATE  ') THEN { regular method }
002280                  BEGIN
002281                  addr := calledA6+8;
002282                  IF ValidObject(Handle(TpLONGINT(addr)^)) THEN
002283                      frameSELF := TObject(TpLONGINT(addr)^);
002284                  END;
002285
002286              IF frameSELF <> NIL THEN
002287                  BEGIN
002288                  LIntToHex(ORD(frameSELF), @hexStr);
002289                  class := frameSELF.Class;
002290                  CpToCn(TPSliceTable(class), TS8(className));
002291                  Write('  (', className, ': $', hexStr, ')');
002292                  END;
002293
002294              IF full THEN
002295                  BEGIN
002296                  {$IFC fDebugMethods}
002297                  WriteLn;
002298                  IF frameSELF <> NIL THEN
002299                      BEGIN
```

```
002300                  Write('SELF = ');
002301                  currXPos := 7;
002302                  outputIndent := 7;
002303                  frameSELF.Debug(1, '');
002304                  WriteLn;
002305                  END;
002306              {$ENDC}
002307
002308              localBytes := Max(0, Min(ORD(calledA6 - (laterA6 + 8)), $50));
002309              paramBytes := Max(0, Min(ORD(earlierA6 - (calledA6 + 8)), $50));
002310              selfBytes := 4 * ORD(frameSELF <> NIL);
002311              WriteLn;
002312              WriteLn('LOCALS (First declared local is listed last):');
002313              WrMemory(calledA6 - localBytes, localBytes, FALSE);
002314              WriteLn('PARAMETERS (Last declared parameter is listed first):');
002315              WrMemory(calledA6 + 8 + selfBytes, paramBytes - selfBytes, FALSE);
002316              END;
002317          END;
002318
002319      WriteLn;
002320  END;
002321
002322  PROCEDURE StackCrawl;
002323      VAR frNum:          INTEGER;
002324  BEGIN
002325      frNum := 1;
002326      WHILE WrFrame(frNum, FALSE) DO
002327          frNum := frNum + 1;
002328      WriteLn;
002329  END;
002330
002331  PROCEDURE FrameDump;
002332      VAR i:      INTEGER;
002333          frame:  LONGINT;
002334  BEGIN
002335      GetToken;
002336      IF token = '' THEN
002337          BEGIN
002338          Write('Frame number to dump? ');
002339          ReadLn(token);
002340          END;
002341      StrToInt(@token,i,cState);
002342      IF cState = cvValid THEN
002343          BEGIN
002344          IF (i >= 1) THEN
002345              IF NOT WrFrame(i, TRUE) THEN
002346                  WriteLn('Frame number was too large');
002347          END;
```

```
002348          WriteLn;
002349      END;
002350
002351      PROCEDURE ToPrinter;
002352          VAR errnum:     INTEGER;
002353              outfname:   PathName;
002354
002355  {$IFC LibraryVersion <= 20}
002356              { Paslib initialization done in the WorkShop that is not done in the DeskTop manager }
002357          PROCEDURE TellPaslibPrinterLocation;
002358              CONST
002359                  AlreadyMounted = 1052;
002360
002361              VAR
002362                  errnum:       INTEGER;                          { error return }
002363                  tp:           TPORTS;
002364                  devname:      E_Name;
002365                  vname:        E_Name;
002366                  password:     E_Name;
002367                  tdt:          TDeviceType;
002368                  tdi:          TDeviceInfo;
002369                  dsp:          DsProcParam;
002370                  DevControl:   DcType;
002371                  path:         PathName;
002372
002373          BEGIN
002374              FOR tp := uppertwig TO t_extra3 DO
002375                  BEGIN
002376                  Get_config_name(errnum,tp,devname);
002377                  IF errnum <= 0 THEN
002378                      BEGIN
002379                      PMReadConfig(tp,tdt,tdi);
002380                      IF tdt IN [DMPrinter,Typer] THEN
002381                          BEGIN
002382                          Mount(errnum, vname, password, devname);
002383                          IF (errnum <= 0) or (errnum = AlreadyMounted) THEN
002384                              BEGIN
002385                              dsp.proccode := dsPrintDev;
002386                              dsp.PrDevice := Concat('-', devname);
002387                              DSPaslibCall(dsp);
002388                              WITH DevControl DO
002389                                  BEGIN
002390                                  path := Concat('-',devname,'-x'); {OD}
002391                                  dcversion := 2;
002392                                  dccode := 17;   {auto LF disable}
002393                                  dcdata[0] := 1;
002394                                  Device_Control(errnum,path,DevControl);
002395                                  CASE tp OF
```

```
002396                                  seriala,serialb:
002397                                    BEGIN
002398                                      dccode := 5;
002399                                      dcdata[0] := 9600;
002400                                      Device_control(errnum,path,devcontrol); {baud rate}
002401                                      dccode := 2;
002402                                      Device_control(errnum,path,devcontrol); {DTR}
002403                                      dccode := 1;
002404                                      dcdata[0] := 0;
002405                                      Device_control(errnum,path,devcontrol); {8-bit no-parity}
002406                                      dccode := 12;
002407                                      dcdata[0] := 60;
002408                                      Device_control(errnum,path,devcontrol); {time out}
002409                                      dccode := 10;
002410                                      dcdata[0] := 0;
002411                                      dcdata[1] := -128;
002412                                      Device_control(errnum,path,devcontrol); {disconnect detect}
002413                                    END;
002414                                END {CASE}
002415                            END;
002416                        EXIT(TellPaslibPrinterLocation);
002417                        END
002418                    ELSE
002419                        WriteLn('Error number ',errnum,' mounting ',devname);
002420                    END;
002421                END
002422            END;
002423        END { TellPaslibPrinterLocation };
002424  {$ELSEC} {Spring Version}
002425            { Paslib initialization done in the WorkShop that is not done in the DeskTop manager }
002426        PROCEDURE TellPaslibPrinterLocation;
002427            CONST
002428                AlreadyMounted = 1052;
002429                cdSerialCable = 32;
002430
002431            TYPE
002432                TExtWords = PACKED RECORD
002433                        isPrinter:  BOOLEAN;
002434                        isDefault:  BOOLEAN;
002435                        driverID:   -8192..8191;
002436                        END;
002437
002438            VAR
002439                errnum:         INTEGER;  { error return }
002440                nextEntry:      LONGINT;
002441                config:         ConfigDev;
002442                pExtWords:      ^TExtWords;
002443                vname:          E_Name;
```

```
002444                password:       E_Name;
002445                devControl:     DcType;
002446                dsp:            DsProcParam;
002447                path:           Pathname;
002448          BEGIN
002449              nextEntry := 0;
002450              REPEAT
002451                  PmReadConfig(errnum, nextEntry, config);
002452                  IF errnum <= 0 THEN
002453                      IF config.nExtWords >= 1 THEN
002454                          BEGIN
002455                          pExtWords := @config.extWords[1];
002456                          IF pExtWords^.isPrinter THEN
002457                              BEGIN
002458                              Mount(errnum, vname, password, config.devname);
002459                              IF (errnum <= 0) or (errnum = AlreadyMounted) THEN
002460                                  BEGIN
002461                                  dsp.proccode := dsPrintDev;
002462                                  dsp.PrDevice := Concat('-', config.devname);
002463                                  DSPaslibCall(dsp);
002464                                  WITH devControl DO
002465                                      BEGIN
002466                                      path := Concat('-', config.devname, '-x'); {OD}
002467                                      dcversion := 2;
002468                                      dccode := 17;   {auto LF disable}
002469                                      dcdata[0] := 1;
002470                                      Device_Control(errnum, path, devControl);
002471
002472                                      IF config.driverID = cdSerialCable THEN
002473                                          BEGIN
002474                                          dccode := 5;
002475                                          dcdata[0] := 9600;
002476                                          Device_control(errnum,path,devcontrol); {baud rate}
002477                                          dccode := 2;
002478                                          Device_control(errnum,path,devcontrol); {DTR}
002479                                          dccode := 1;
002480                                          dcdata[0] := 0;
002481                                          Device_control(errnum,path,devcontrol); {8-bit no-parity}
002482                                          dccode := 12;
002483                                          dcdata[0] := 60;
002484                                          Device_control(errnum,path,devcontrol); {time out}
002485                                          dccode := 10;
002486                                          dcdata[0] := 0;
002487                                          dcdata[1] := -128;
002488                                          Device_control(errnum,path,devcontrol); {disconnect detect}
002489                                          END; {IF config.driverID = cdSerialCable}
002490                                      END; {WITH devControl DO}
002491                                  END {IF (errnum <= 0) or (errnum = AlreadyMounted) THEN}
```

```
002492                          ELSE
002493                              WriteLn('Error number ', errnum, ' mounting ', config.devname);
002494                          END; {IF pExtWords^.isPrinter THEN}
002495                      END; {IF config.nExtWords >= 1 THEN}
002496              UNTIL errnum > 0;
002497          END { TellPaslibPrinterLocation };
002498  {$ENDC}
002499
002500      BEGIN
002501          GetToken;
002502          outfname := token;
002503          IF token = '' THEN
002504              BEGIN
002505              Write('Name of file to send output to? [-console] ');
002506              ReadLn(outfname);
002507              END;
002508          IF outfname = '' THEN
002509            OutputRedirect(errnum,outfname,TRUE)
002510          ELSE
002511            BEGIN
002512              StrUpperCased(@outfname);
002513              IF outfname = '-PRINTER' THEN
002514                TellPaslibPrinterLocation;
002515              OutputRedirect(errnum,outfname,FALSE);
002516            END;
002517          IF errnum > 0 THEN
002518              BEGIN
002519              IF outfname = '' THEN
002520                  outfname := '-CONSOLE';
002521              WriteLn('Error number ',errnum,' redirecting output to ',outfname);
002522              END;
002523      END;
002524
002525      PROCEDURE MemoryDump;
002526          VAR start:      LONGINT;
002527              numBytes:   LONGINT;
002528      BEGIN
002529          GetToken;
002530          IF token = '' THEN
002531              BEGIN
002532              Write('Starting address [# bytes]? ');
002533              Readln(inputStr);
002534              GetToken;
002535              END;
002536
002537          HexStrToLInt(@token, start, cState);
002538          IF cState <> cvValid THEN
002539              Exit(MemoryDump)
```

```
002540          ELSE
002541              GetToken;
002542
002543          HexStrToLInt(@token, numBytes, cState);
002544          IF cState <> cvValid THEN
002545              numBytes := $10;
002546
002547          WrMemory(start, numBytes, TRUE);
002548      END;
002549
002550      PROCEDURE HeapDump;
002551          VAR allInfo:        BOOLEAN;
002552              wantedSTP:      LONGINT; {-1 for all classes}
002553              allHeaps:       BOOLEAN;
002554              index:          INTEGER;
002555              heap:           THeap;
002556              dumpDocHeap:    BOOLEAN;
002557      BEGIN
002558          allInfo := TRUE;
002559          wantedSTP := -1;
002560          allHeaps := TRUE;
002561
002562          GetToken;
002563          IF token <> '' THEN
002564              BEGIN
002565              allHeaps := FALSE;
002566              TrimBlanks(@token);
002567              StrUpperCased(@token);
002568              index := CiOfCn(Copy(Concat(token, '       '), 1, 8));
002569              IF index > 0 THEN
002570                  wantedSTP := ORD(hMySTables^^.records[index])
002571              ELSE
002572                  BEGIN
002573                  WriteLn('No such class!');
002574                  EXIT(HeapDump);
002575                  END;
002576              END;
002577
002578          IF allHeaps THEN
002579              IF NOT YesNo('All classes') THEN
002580                  WHILE wantedSTP <= 0 DO
002581                      BEGIN
002582                      GetOne('Which class?');
002583                      IF token = '' THEN
002584                          GOTO 99
002585                      ELSE
002586                          BEGIN
002587                          index := CiOfCn(token);
```

```
002588                      IF index > 0 THEN
002589                          wantedSTP := ORD(hMySTables^^.records[index])
002590                      ELSE
002591                          WriteLn('No such class!');
002592                      END;
002593                  END;
002594
002595          {$IFC fDebugMethods}
002596          IF wantedSTP <= 0 THEN
002597              allInfo := YesNo('Dump fields as well as class of each object');
002598          {$ELSEC}
002599          allInfo := FALSE;
002600          {$ENDC}
002601
002602          IF allHeaps THEN
002603              IF YesNo('Dump Process Heap') THEN
002604                  DumpHeap(mainHeap, wantedSTP, 0, allInfo);
002605
002606          IF isInitialized THEN
002607              BEGIN
002608              IF allHeaps THEN
002609                  dumpDocHeap := YesNo('Dump Active Document Heap')
002610              ELSE
002611                  dumpDocHeap := TRUE;
002612
002613              IF dumpDocHeap THEN
002614                  BEGIN
002615                  heap := BindHeap(TRUE, TRUE);
002616                  IF heap = NIL THEN
002617                      BEGIN
002618                      WriteLn('There is no active document heap!');
002619                      WriteLn;
002620                      WriteLn;
002621                      END
002622                  ELSE
002623                      DumpHeap(heap, wantedSTP, 0, allInfo);
002624                  END;
002625
002626              IF allHeaps THEN
002627                  IF YesNo('Dump Clipboard Heap') THEN
002628                      BEGIN
002629                      heap := BindHeap(FALSE, TRUE);
002630                      IF heap = NIL THEN
002631                         BEGIN
002632                         WriteLn('There is no clipboard heap!  Maybe no cut/copy has been done since booting.');
002633                         WriteLn;
002634                         END
002635                      ELSE
```

```
002636                      DumpHeap(heap, wantedSTP, 0, allInfo);
002637                  heap := BindHeap(FALSE, FALSE);
002638                  END;
002639              END
002640          ELSE
002641              BEGIN
002642              WriteLn('There is no document or clipboard heap because the process is not fully initialized');
002643              WriteLn;
002644              END;
002645      END;
002646
002647  BEGIN {EntDebugger}
002648      SetScreenKeybd(altscrn);
002649      traceCount := defTraceCount;
002650      timeToGo := FALSE;
002651      WriteLn;
002652      WriteLn('ToolKit Debugger - ',enterReason);
002653
002654      { flush characters; because of PASLIB bug, also stop when user types a ~ }
002655  (*  commented out as a test case to see if still needed here
002656      IF CheckKeyPress('') THEN;
002657  *)
002658
002659      IF tallyingCalls THEN
002660          BEGIN
002661          TallyAndTime;
002662          returnToMain := tallyingCalls;
002663          END;
002664
002665      IF NOT tallyingCalls THEN
002666          BEGIN
002667          REPEAT
002668              IF NOT timeToGo THEN
002669                  BEGIN
002670                  IF showPrompt THEN
002671                     BEGIN
002672                     WriteLn('B)reakpoint, C)learBreakpoints [breakpoint #/ALL], D)ebugStatus, E)nterLisabug,');
002673                     {$IFC fDebugMethods}
002674                     WriteLn('     F)rameDump, G)o, H)eapDump [class], I)nspectObject, L)evelsToWatch,');
002675                     {$ELSEC}
002676                     WriteLn('     F)rameDump, G)o, H)eapDump [class], L)evelsToWatch,');
002677                     {$ENDC}
002678                     WriteLn('     M)emoryDump <location> [# bytes], O)utputTo, P)rompt [Y/N], R)efsToObject,');
002679                     WriteLn('     S)tackCrawl, T)ally & Time, W)atch [count] [A)ltScreen] [C)lass] [F)ields]');
002680                     END;
002681                  Write('-->');
002682                  ReadLn(inputStr);
002683                  END;
```

```
002684              GetToken;
002685              IF token <> '' THEN
002686                  CASE CharUpperCased(token[1]) OF
002687                      null:   GoKitBug;   {don't expect people to type this command, but this
002688                                           will guarantee that the Linker does not flush
002689                                           the GoKitBug procedure}
002690                      'B':    BrSetup('Break on');
002691                      'C':    ClearBreaks;
002692                      'D':    DebugStatus;
002693                      'E':    %_GoLisabug;
002694                      'F':    FrameDump;
002695                      'G':    BEGIN
002696                              fTraceEnabled := FALSE;
002697                              defTraceCount := 0;
002698                              traceCount := defTraceCount;
002699                              returnToMain := TRUE;
002700                              timeToGo := TRUE;
002701                              END;
002702
002703                      'H':    HeapDump;
002704                      {$IFC fDebugMethods}
002705                      'I':    Inspect;
002706                      {$ENDC}
002707                      'L':    Level;
002708                      'M':    MemoryDump;
002709                      'O':    ToPrinter;
002710                      'P':    PromptOnOff;
002711                      'R':    RefsToObject;
002712                      'S':    StackCrawl;
002713                      'T':    BEGIN
002714                              TallyAndTime;
002715                              timeToGo := tallyingCalls;
002716                              END;
002717
002718                      'W':    BEGIN
002719                              TraceOrNot;
002720                              timeToGo := TRUE;
002721                              END;
002722
002723                      'X':    timeToGo := TRUE;
002724                      END;
002725              99:
002726            UNTIL timeToGo;
002727          END;
002728
002729      IF tallyingCalls THEN
002730          BEGIN
002731          fTraceEnabled := FALSE;
```

```
002732          defTraceCount := 0;
002733          traceCount := defTraceCount;
002734          returnToMain := TRUE;
002735          END;
002736
002737      IF returnToMain THEN
002738          SetScreenKeybd(priscrn);
002739
002740  END; {EntDebugger}
002741  {$S SgCLAdbg}
002742
002743
002744  FUNCTION  AKeyPress: BOOLEAN;
002745      VAR tb: BOOLEAN;
002746  BEGIN
002747      AKeyPress := FALSE;
002748      IF kpcntr >= keyPresLimit THEN
002749          BEGIN
002750          tb := KeyPress;
002751              { force call to keyPress until press is dealt with }
002752          IF NOT tb THEN
002753              kpcntr := 0;
002754          AKeyPress := tb;
002755          END
002756      ELSE
002757          kpcntr := kpcntr + 1
002758  END;
002759
002760
002761  { ==================================== BP -- EP ======================================== }
002762
002763
002764  {$IFC fTrace}
002765  PROCEDURE BEPSN(odummy: LONGINT; fBegin, displayIt: BOOLEAN);
002766      VAR receiver:       TObject;
002767          caller:         TpLONGINT;
002768          i:              INTEGER;
002769          className:      TClassName;
002770          procName:       S8;
002771          toDebugger:     BOOLEAN;
002772          nextPC:         LONGINT;
002773
002774        { See if this is the method to start tracing at }
002775      PROCEDURE BreakHere;
002776          VAR ts:     S16;
002777              i:      INTEGER;
002778              found:  BOOLEAN;
002779      BEGIN
```

```
002780          found := FALSE;
002781          FOR i := 1 TO breakMCount DO
002782              BEGIN
002783              WITH breakMethods[i] DO
002784                  { NOTE: if both brClass and brMethod are '', then the iTH breakpoint is unassigned }
002785                  IF brClass = '' THEN
002786                      found :=  (brMethod = procName) AND (brMethod <> '')
002787                  ELSE
002788                  IF brMethod = '' THEN
002789                      found := brClass = className
002790                  ELSE
002791                      found := (brClass = className) AND (brMethod = procName);
002792              IF found THEN
002793                  BEGIN
002794                  displayIt := TRUE;
002795                  toDebugger := TRUE;
002796                  returnToMain := TRUE;
002797                  fTraceSelf := FALSE;
002798                  fTraceClass := TRUE;
002799                  lastBpPc := 0;
002800                  lastEpPc := 0;
002801                  EXIT(BreakHere)
002802                  END
002803              END
002804      END;
002805
002806      PROCEDURE WriteOutDebugInfo;
002807          CONST maxIndent = 70;
002808          VAR i:      INTEGER;
002809              hexStr: S8;
002810      BEGIN
002811          WriteLn;
002812          indentTrace := CMin(tabLevel, maxIndent + 5);
002813
002814          IF tabLevel <= trLevMemory THEN
002815              Write(traceLevels[tablevel]:4, ' ')
002816          ELSE
002817              Write('     ');
002818
002819          Write(' ': CMin(tabLevel, maxIndent));
002820
002821          IF tabLevel > maxIndent + 5 THEN
002822              Write(tabLevel:4, ' ')
002823          ELSE
002824          IF indentTrace > maxIndent THEN
002825              Write(' ': indentTrace - maxIndent);
002826
002827          IF fBegin THEN
```

```
002828              Write('BEGIN ')
002829          ELSE
002830              Write('END   ');
002831          {$IFC fDebugMethods}
002832          currXPos := indentTrace + 11 {5 for level #; 6 for BEGIN/END};
002833          {$ENDC}
002834
002835          IF className<>'' THEN
002836              BEGIN
002837              Write(className, '.');
002838              {$IFC fDebugMethods}
002839              currXpos := currXPos + 9;
002840              {$ENDC}
002841              END;
002842          Write(procName);
002843          {$IFC fDebugMethods}
002844          currXPos := currXPos + 8;
002845          {$ENDC}
002846
002847          IF (fTraceSelf OR fTraceClass) AND (receiver <> NIL) THEN
002848              IF (procName<>'DEBUGOBJ') AND (procName<>'DEBUG   ') AND (procName<>'FIELDS  ') AND
002849                 (fBegin OR ((procName<>'FREEOBJE') AND (procName<>'FREE    '))) THEN
002850                  BEGIN
002851                  {$IFC fDebugMethods}
002852                  Write('(');
002853                  currXPos := currXPos + 1;
002854
002855                  IF (procName <> 'FREEOBJE') AND fTraceSelf THEN
002856                      BEGIN
002857                      outputIndent := currXPos;
002858                      receiver.Debug(1, '');
002859                      END
002860                  ELSE
002861                      BEGIN
002862                      receiver.Debug(0, '');
002863                      LIntToHex(LONGINT(receiver), @hexStr);
002864                      Write(': $', hexStr);
002865                      END;
002866
002867                  Write(')');
002868                  {$ENDC}
002869                  END;
002870      END;
002871
002872      PROCEDURE TraceStuff;
002873          VAR nextPC: LONGINT;
002874      BEGIN
002875          IF traceCount = 1 THEN
```

```
002876              EntDebugger(' ','Count methods displayed')
002877          ELSE
002878              traceCount := traceCount - 1
002879      END;
002880
002881  BEGIN
002882      toDebugger := FALSE;
002883
002884      IF fDebugRecursion THEN
002885          EXIT(BEPSN);
002886      fDebugRecursion := TRUE;
002887
002888      caller := TpLONGINT(odummy + 4);
002889      receiver := NIL;
002890      IF GetDollarD(TppINTEGER(caller), className, procName, nextPC) THEN
002891          IF (className <> '') AND (procName <> 'CREATE  ') THEN
002892              IF ValidObject(Handle(TpLONGINT(caller^+8)^)) THEN
002893                  receiver := TObject(TpLONGINT(caller^+8)^);
002894
002895      IF breakMCount > 0 THEN
002896          BreakHere;
002897
002898      IF displayIt THEN
002899          WriteOutDebugInfo;
002900
002901      IF toDebugger THEN
002902          EntDebugger(' ','Breakpoint found')
002903      ELSE
002904          BEGIN
002905          IF displayIt THEN
002906              toDebugger := KeyPress
002907          ELSE
002908              toDebugger := AKeyPress;
002909
002910          IF toDebugger THEN
002911              EntDebugger(' ','Key pressed on alternate screen')
002912          ELSE
002913          IF (traceCount > 0) and (displayIt) THEN
002914              TraceStuff;
002915          END;
002916
002917      fDebugRecursion := FALSE;
002918  END;
002919
002920
002921  PROCEDURE BP{(myTraceLevel: INTEGER)};
002922      VAR dummy:      LONGINT;    {Must be first VAR}
002923          bpFrame:    TpLONGINT;
```

```
002924          callerPC:   LONGINT;
002925          departed:   LONGINT;
002926  BEGIN
002927      IF tallyingCalls THEN
002928          stopTime := MicroTimer(* - debugTime*);
002929
002930      tabLevel := tabLevel + 1;  {Increment first because BEPSN can be reentrant}
002931      callerPC := TpLONGINT(ORD(@dummy) + 8)^;
002932      IF tabLevel <= trLevMemory THEN
002933          BEGIN
002934          traceLevels[tabLevel] := myTraceLevel;
002935          bpFrame := TpLONGINT(ORD(@dummy) + 4);
002936          traceFrames[tabLevel] := LIntAndLInt(bpFrame^, $00FFFFFF);
002937          END;
002938
002939      IF fTraceEnabled AND (myTraceLevel >= curTraceLevel) THEN
002940          BEPSN(ORD(@dummy), TRUE, TRUE)
002941      ELSE
002942      IF (breakMCount > 0) OR AKeyPress THEN
002943          IF callerPC <> lastBpPc THEN
002944              BEPSN(ORD(@dummy), TRUE, FALSE);
002945      lastBpPc := callerPC;
002946
002947      IF tallyingCalls THEN
002948          BEGIN
002949          departed := MicroTimer;
002950  (*      debugTime := debugTime + departed - stopTime + tallyOverhead; *)
002951          IF tabLevel <= trLevMemory THEN
002952              traceTimes[tabLevel] := departed (*- debugTime*);
002953          END;
002954  END;
002955
002956
002957  PROCEDURE EP;
002958      VAR dummy:      LONGINT;    {Must be first VAR and 4 bytes long}
002959          epFrame:    LONGINT;
002960          doTrace:    BOOLEAN;
002961          i:          INTEGER;
002962          callerPC:   LONGINT;
002963          elapsed:    LONGINT;
002964  BEGIN
002965      callerPC := TpLONGINT(ORD(@dummy) + 8)^;
002966
002967      IF tallyingCalls THEN
002968          BEGIN
002969          stopTime := MicroTimer (*- debugTime*);
002970
002971          IF tabLevel <= trLevMemory THEN
```

```
002972              BEGIN
002973              elapsed := stopTime - traceTimes[tabLevel];
002974              FOR i := tabLevel - 1 DOWNTO 1 DO
002975                  traceTimes[i] := traceTimes[i] + elapsed;
002976              Tally(callerPC, elapsed);
002977              END
002978          ELSE
002979              BEGIN
002980              WriteLn('Stack bigger than performance measurement can handle! ', tablevel:1);
002981              tallyingCalls := FALSE;
002982              END;
002983          END;
002984
002985      IF tabLevel < 0 THEN
002986          BEGIN
002987          tabLevel := 0;
002988          Writeln('-------------------------');
002989          BEPSN(ORD(@dummy), FALSE, TRUE);
002990          ABCBreak('The above EP had no BP at all', 0);
002991          doTrace := FALSE;
002992          END
002993      ELSE IF tabLevel <= trLevMemory THEN
002994          BEGIN
002995          epFrame := LIntAndLInt(TpLONGINT(ORD(@dummy) + 4)^, $00FFFFFF);
002996          IF traceFrames[tabLevel] <> epFrame THEN
002997              BEGIN
002998              i := tabLevel - 1;      {Try to resynchronize}
002999              WHILE (tabLevel <> i) AND (i >= 0) DO
003000                  IF traceFrames[i] = epFrame THEN
003001                      BEGIN
003002                      Writeln('-------------------------');
003003                      ABCBreak('There was a BP with no EP', 0);
003004                      tabLevel := i;
003005                      END
003006                  ELSE
003007                      i := i - 1;
003008              IF tabLevel <> i THEN
003009                  BEGIN
003010                  Writeln('-------------------------');
003011                  BEPSN(ORD(@dummy), FALSE, TRUE);
003012                  ABCBreak('The above EP had no BP', 0);
003013                  END;
003014              END;
003015          doTrace := fTraceEnabled AND (traceLevels[tablevel] >= curTraceLevel);
003016          END
003017      ELSE
003018          doTrace := FALSE;
003019
```

```
003020      IF doTrace THEN
003021          BEPSN(ORD(@dummy), FALSE, TRUE)
003022      ELSE
003023      IF (breakMCount > 0) OR AKeyPress THEN
003024          IF callerPC <> lastEpPc THEN
003025              BEPSN(ORD(@dummy), FALSE, FALSE);
003026
003027      IF tabLevel >= 0 THEN
003028          tabLevel := tabLevel - 1;
003029      lastEpPc := callerPC;
003030
003031  (*  IF tallyingCalls THEN
003032          debugTime := MicroTimer - stopTime + tallyOverhead;
003033      *)
003034  END;
003035  {$ENDC}
003036  {$ENDC}
003037
003038
003039  { ================================== COUNTHEAP ================================== }
003040
003041
003042  {$IFC fCheckHeap}
003043  FUNCTION CountHeap{(heap: THeap): INTEGER};
003044      VAR hz:         THz;
003045          numObjects: INTEGER;
003046  BEGIN
003047      hz := THz(heap);
003048      IF FCheckHzOK(hz, numObjects) THEN ;
003049      CountHeap := numObjects;
003050  END;
003051  {$ENDC}
003052
003053
003054  {$S sInit1}
003055
003056
003057
```

End of File -- Lines: 3057 Characters: 100113

```
================================================================================================
FILE: "LIBTK/UTEXT.TEXT"
================================================================================================

000001  UNIT UText;
000002  {$SETC IsIntrinsic := TRUE }
000003
000004  {$IFC IsIntrinsic}
000005  INTRINSIC;
000006  {$ENDC}
000007
000008
000009  {Multiple Paragraph Building Block for the Tool Kit}
000010
000011  {changed 04/25/84 1437  Added TTextImage.TxtImgForClipBoard method}
000012  {changed 04/18/84 1652  Added firstLinePixel, useFirstPixel fields to TTextImage}
000013  {changed 04/16/84 1135  Added styleSheet field to TParaFormat}
000014  {changed 04/13/84 0209  Added TTextImage.NewEditPara}
000015  {changed 04/12/84 2344  Changed parameter list of TParagraph.UpdateRuns}
000016  {changed 04/10/84 1400  Changed TEditPara.images field back to a TList}
000017
000018  INTERFACE
000019  {$DECL fUseUnivText}
000020  {$SETC fUseUnivText := TRUE}
000021
000022  USES
000023      {$U libtk/UObject}            UObject,
000024  {$IFC LibraryVersion <= 20}
000025      {$U UFont}              UFont,
000026  {$ENDC}
000027      {$U QuickDraw}          QuickDraw,
000028      {$U libtk/UDraw}              UDraw,
000029  {$IFC fUseUnivText}
000030      {$U libtk/UUnivText}          UTKUniversalText,
000031  {$ENDC}
000032      {$U UABC}               UABC;
000033
000034  {$DECL fTextTrace}
000035  {$SETC fTextTrace := fDbgOK}
000036  {$DECL fParaTrace}
000037  {$SETC fParaTrace := fDbgOK}
000038  {$DECL fRngText}
000039  {$SETC fRngText := fDbgOK}
000040
000041  CONST
000042
000043      cVertMargin = 4;
```

```
000044      cHorizMargin = 6;
000045
000046      somethingKind = 1;
000047
000048  TYPE
000049
000050      TStyleChange = RECORD
000051          lp:         INTEGER;
000052          newStyle:   TTypeStyle;
000053          END;
000054
000055      TTxtTabDescriptor = RECORD
000056          xCoord:     INTEGER;
000057          quad:       TAlignment;
000058          {MORE LATER}
000059          END;
000060
000061      TDrawAction = (actionDraw, actionInval, actionNone);
000062
000063  { PARAGRAPH SUBCLASSES }
000064
000065      TParaFormat = SUBCLASS OF TObject
000066          dfltTStyle:         TTypeStyle;      {default type style}
000067          wordWrap:           BOOLEAN;
000068          quad:               TAlignment;
000069          firstIndent:        INTEGER;
000070          leftIndent:         INTEGER;
000071          rightIndent:        INTEGER;
000072          spaceAbovePara:     INTEGER;
000073          spaceBelowPara:     INTEGER;
000074          lineSpacing:        INTEGER;
000075          tabs:               TArray;
000076
000077          refCount:           INTEGER;        {number of paragraphs referencing this paraFormat}
000078          permanent:          BOOLEAN;        {TRUE -> don't free when refcount goes to zero}
000079          styleSheet:         TStyleSheet;    {NIL if format not in a styleSheet}
000080
000081          FUNCTION TParaFormat.CREATE(object: TObject; heap: THeap; itsStyleSheet: TStyleSheet): TParaFormat;
000082          {$IFC fParaTrace}
000083          PROCEDURE TParaFormat.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000084          {$ENDC}
000085          PROCEDURE TParaFormat.SetTypeStyle(tStyle: TTypeStyle);
000086          PROCEDURE TParaFormat.ChangeRefCountBy(delta: INTEGER);
000087          END;
000088
000089      TParagraph = SUBCLASS OF TString
000090          typeStyles:     TArray; { of TStyleChange }
000091
```

```
000092        {Creation/Destruction}
000093          FUNCTION  TParagraph.CREATE(object: TObject; heap: THeap;
000094                                       initialSize: INTEGER; initialTypeStyle: TTypeStyle): TParagraph;
000095          PROCEDURE TParagraph.Free; OVERRIDE;
000096
000097        {Debugging}
000098          {$IFC fParaTrace}
000099          PROCEDURE  TParagraph.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000100          {$ENDC}
000101
000102        {Overridden TString methods}
000103          PROCEDURE TParagraph.Draw(i: LONGINT; howMany: INTEGER); OVERRIDE;
000104          FUNCTION  TParagraph.Width(i: LONGINT; howMany: INTEGER): INTEGER; OVERRIDE;
000105
000106        {This method is used by TParagraph.Draw and TParagraph.Width to interpret the typeStyles array}
000107          PROCEDURE TParagraph.DrawLine(startLP, endLP: INTEGER; fDraw: BOOLEAN; fWidth: BOOLEAN;
000108                                        VAR width: INTEGER; VAR styleIndex: INTEGER);
000109
000110        {Type Style Maintainence}
000111          PROCEDURE TParagraph.ChangeStyle(startLP, endLP: INTEGER; PROCEDURE Change(VAR typeStyle: TTypeStyle);
000112                                                            VAR styleOfStartLP: TTypeStyle);
000113
000114
000115          {These four routines all call ChangeStyle}
000116          PROCEDURE TParagraph.ChgFace(startLP, endLP: INTEGER;
000117                                       newOnFaces: {$IFC LibraryVersion <= 20}TSeteface{$ELSEC}Style{$ENDC};
000118                                                            VAR styleOfStartLP: TTypeStyle);
000119          PROCEDURE TParagraph.ChgFontSize(startLP, endLP: INTEGER; newFontSize: Byte;
000120                                                            VAR styleOfStartLP: TTypeStyle);
000121          PROCEDURE TParagraph.ChgFontFamily(startLP, endLP: INTEGER; newFontFamily: Byte;
000122                                                            VAR styleOfStartLP: TTypeStyle);
000123          PROCEDURE TParagraph.NewStyle(startLP, endLP: INTEGER; newTypeStyle: TTypeStyle);
000124
000125          PROCEDURE TParagraph.CleanRuns;
000126          PROCEDURE TParagraph.UpdateRuns(atLP: INTEGER; replacedChars: INTEGER; insertedChars: INTEGER);
000127
000128        {Character Maintainence}
000129          PROCEDURE TParagraph.ReplPara(fPos, numChars: INTEGER;
000130                                        otherPara: TParagraph; otherFPos, otherNumChars: INTEGER);
000131          PROCEDURE TParagraph.ReplTString(fPos, numChars: INTEGER;
000132                                        otherString: TString; otherFPos, otherNumChars: INTEGER);
000133          PROCEDURE TParagraph.ReplPString(fPos, numChars: INTEGER; pStr: TPString);
000134        {Utilities}
000135        {BuildExtentLRect takes an LPoint that indicates the baseline of the paragraph.  It returns
000136         in extentLRect the bounding rectangle whose height is based on the tallest font in the
000137         paragraph and width is the width of the characters in the paragraph. Specifically:
000138                         top    := baseLPt.v - tallestFontInfo.ascent;
000139                         bottom := baseLPt.v + tallestFontInfo.descent + tallestFontInfo.leading;
```

```
000140                        left   := baseLPt.h;
000141                        right  := baseLpt.h + paragraph.Width;}
000142          PROCEDURE TParagraph.BuildExtentLRect(baseLPt: LPoint; VAR extentLRect: LRect);
000143          FUNCTION  TParagraph.FixLP(LP: INTEGER): INTEGER;
000144          PROCEDURE TParagraph.SetTypeStyle(tStyle: TTypeStyle);
000145          PROCEDURE TParagraph.StyleAt(lp: INTEGER; VAR typeStyle: TTypeStyle);
000146
000147        {Word Selection}
000148          PROCEDURE TParagraph.FindWordBounds(orig: INTEGER; VAR first, last: INTEGER);
000149          FUNCTION  TParagraph.Qualifies(pos: INTEGER): BOOLEAN;
000150
000151          END;
000152
000153
000154      {Editable Paragraph}
000155       TEditPara = SUBCLASS OF TParagraph
000156         { character stuff }
000157           bsCount:        INTEGER;
000158         { formatting stuff }
000159           nestLevel:      INTEGER;
000160           format:         TParaFormat;
000161
000162         { paraImage stuff }
000163           beingFiltered:  BOOLEAN;    { TRUE when a type style command has just been
000164                                         performed on this paragraph}
000165
000166           (*
000167           maxImage:       INTEGER;
000168           numImages:      INTEGER;
000169           images:         ARRAY [1..1] OF TParaImage;  {THIS MUST BE LAST FIELD !}
000170           *)
000171           images:         TList;      { Users may subclass TEditPara }
000172
000173         {Creation/Destruction}
000174           FUNCTION  TEditPara.CREATE(object: TObject; heap: THeap; initialSize: INTEGER;
000175                                      itsFormat: TParaFormat): TEditPara;
000176           PROCEDURE TEditPara.Free; OVERRIDE;
000177
000178         {Debugging}
000179           {$IFC fParaTrace}
000180           PROCEDURE TEditPara.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000181           {$ENDC}
000182
000183         {Special Editing}
000184           PROCEDURE TEditPara.BeginInsertion(atLP: INTEGER; size:INTEGER);
000185           PROCEDURE TEditPara.EndInsertion;
000186           FUNCTION  TEditPara.GrowSize: INTEGER;
000187           PROCEDURE TEditPara.InsertOneChar(ch: CHAR; atLP: INTEGER);
```

```
000188
000189        {Utility}
000190          PROCEDURE TEditPara.SetTypeStyle(tStyle: TTypeStyle); OVERRIDE;
000191
000192        {ParaImage Maintenance}
000193          PROCEDURE TEditPara.EachImage(PROCEDURE ImageProc(paraImage: TParaImage));
000194          PROCEDURE TEditPara.DelImage(delImage: TParaImage; fFree: BOOLEAN);
000195          PROCEDURE TEditPara.InsImage(paraImage: TParaImage);
000196          PROCEDURE TEditPara.DelImgIF(FUNCTION ShouldDelete(paraImage: TParaImage): BOOLEAN);
000197          END;
000198
000199
000200      TLineInfo = SUBCLASS OF TObject
000201          valid:          BOOLEAN;
000202          startLP:        INTEGER;
000203          lastDrawnLP:    INTEGER;  {last character in line to draw: may omit trailing spaces}
000204          endLP:          INTEGER;  {last character in line: equals next lineInfo.startLP - 1}
000205          lineLRect:      LRect;
000206          lineAscent:     INTEGER;
000207
000208          FUNCTION  TLineInfo.CREATE(object: TObject; heap: THeap): TLineInfo;
000209          {$IFC fParaTrace}
000210          PROCEDURE TLineInfo.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000211          {$ENDC}
000212
000213        {Used by subclassers who don't like the way the hilite/update
000214         rectangle is chosen so they can overrride it}
000215          FUNCTION TLineInfo.LeftCoord(proposedLeftPixel: LONGINT): LONGINT;
000216          FUNCTION TLineInfo.RightCoord(proposedRightPixel: LONGINT): LONGINT;
000217          END;
000218
000219
000220      TParaImage = SUBCLASS OF TImage
000221          paragraph:      TEditPara;
000222          height:         INTEGER;    { pixel height of the paragraph}
000223
000224          lineList:       TList;      { of TLineInfo}
000225          changed:        BOOLEAN;
000226          tickCount:      INTEGER;    { incremented (mod MAXINT) every time image is drawn }
000227          startLP:        INTEGER;
000228          endLP:          INTEGER;    { while drawing, this is the LP of the beginning of the next line
000229                                        which, when drawing is finished, may be in another image if the
000230                                        paragraph is split }
000231          textImage:      TTextImage; { the textImage that this image belongs to }
000232          wasOffset:      BOOLEAN;    { used by Building block to determine when to invalidate}
000233
000234        {Creation}
000235          FUNCTION  TParaImage.CREATE(object: TObject; heap: THeap; itsView: TView;
```

```
000236                                      itsParagraph: TEditPara; itsLRect: LRect;
000237                                      lineTop: LONGINT; lineLeft: LONGINT): TParaImage;
000238          PROCEDURE TParaImage.Free; OVERRIDE;
000239
000240        {Debugging}
000241          {$IFC fParaTrace}
000242          PROCEDURE TParaImage.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000243          {$ENDC}
000244
000245        {Routines}
000246          PROCEDURE TParaImage.ComputeLineInfo(curLine: TLineInfo; maxLineLen: INTEGER;
000247                                               VAR nextLP: INTEGER; VAR lRectNeeded: LRect);
000248          FUNCTION  TParaImage.DfltLineInfo(lineTop: LONGINT; lineLeft: LONGINT): TLineInfo;
000249          PROCEDURE TParaImage.DrawLine(startLP: INTEGER; fDraw: BOOLEAN;
000250                                        stopWidth, wrapWidth: INTEGER;
000251                                        VAR lineWidth, lastToDraw, endLP: INTEGER);
000252          PROCEDURE TParaImage.DrawParaImage(limitLRect: LRect; startLP: INTEGER; drawAction: TDrawAction;
000253                                             invalBits: BOOLEAN; VAR drawnLRect: LRect);
000254          PROCEDURE TParaImage.Draw; OVERRIDE;
000255          PROCEDURE TParaImage.FastDrawLine(startLP, endLP: INTEGER; fDraw: BOOLEAN; fWidth: BOOLEAN;
000256                                            VAR width: INTEGER; VAR styleIndex: INTEGER);
000257          FUNCTION  TParaImage.GetFormat: TParaFormat;
000258          PROCEDURE TParaImage.LineWithLPt(pt: LPoint; VAR lineIndex: INTEGER; VAR lineInfo: TLineInfo);
000259          PROCEDURE TParaImage.LocateLP(LP: INTEGER; VAR lineIndex: INTEGER; VAR pixel: LONGINT);
000260          FUNCTION  TParaImage.LpWithLPt(pt: LPoint): INTEGER;
000261          PROCEDURE TParaImage.OffSetBy(deltaLPt: LPoint); OVERRIDE;
000262          FUNCTION  TParaImage.ParaTextWidth(startLP, endLP: INTEGER): INTEGER;
000263          PROCEDURE TParaImage.RedrawLines(startLine: INTEGER; endLine: INTEGER);
000264          FUNCTION  TParaImage.SeesSameAs(image: TImage): BOOLEAN; OVERRIDE;
000265
000266         {validation/invalidation procs}
000267          PROCEDURE TParaImage.InvalLinesWith(startLP, endLP: INTEGER);
000268          PROCEDURE TParaImage.AdjustLineLPs(atLP, deltaLP: INTEGER);
000269          END;
000270
000271
000272
000273  { MULTI-PARAGRAPH SUBCLASSES }
000274
000275      TStyleSheet = SUBCLASS OF TObject
000276          formats:    TList; {of TParaFormat}
000277
000278        {Creation}
000279          FUNCTION  TStyleSheet.CREATE(object: TObject; heap: THeap): TStyleSheet;
000280          PROCEDURE TStyleSheet.Free; OVERRIDE;
000281
000282        {Installs Default paraFormat into formats list}
000283          PROCEDURE TStyleSheet.InitDefault;
```

```
000284        {Debugging}
000285          {$IFC fParaTrace}
000286          PROCEDURE TStyleSheet.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000287          {$ENDC}
000288          END;
000289
000290      TTextRange = SUBCLASS OF TObject
000291          firstPara:  TEditPara;
000292          firstIndex: LONGINT;
000293          firstLP:    INTEGER;
000294          lastPara:   TEditPara;
000295          lastIndex:  LONGINT;
000296          lastLP:     INTEGER;
000297
000298        {Creation}
000299          FUNCTION  TTextRange.CREATE(object: TObject; heap: THeap;
000300                                      beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
000301                                      endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER): TTextRange;
000302
000303        {Debugging}
000304          {$IFC fParaTrace}
000305          PROCEDURE TTextRange.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000306          {$ENDC}
000307
000308        {AdjustBy adjust the fields of TTextRange by the value of delta (where delta is in LPs)}
000309          PROCEDURE TTextRange.AdjustBy(delta: INTEGER);
000310          END;
000311
000312      TText = SUBCLASS OF TObject
000313          paragraphs: TList;  {of TEditPara }
000314          styleSheet: TStyleSheet;
000315
000316          txtImgList: TList;  {of TTextImages that point to this text;
000317                               IMPORTANT: If the multiple linked textImage feature is used as described in
000318                                          TTextImage below, the application should only store the
000319                                          head text image in this list.  This list is intended for
000320                                          textImages that are viewing the same text object independently
000321                                          (ie in different panels)}
000322
000323        {Creation/Freeing}
000324          FUNCTION  TText.CREATE(object: TObject; heap: THeap; itsStyleSheet: TStyleSheet): TText;
000325
000326          {DfltTextImage can be called after CREATE to create and return a single textImage. It also
000327           creates one empty paragraph using the first paraFormat in SELF.styleSheet.  It installs the
000328           textImage in txtImgList and the paragraph in paragraphs.  This routine calls
000329           textImage.RecomputeImages to set up the first paraImage.}
000330          FUNCTION  TText.DfltTextImage(view: TView; imageLRect: LRect; imgIsGrowable: BOOLEAN): TTextImage;
000331
```

```
000332          {TText.Free frees all paragraphs that belong to this text object and all textImages that
000333           reference this text object}
000334           PROCEDURE TText.Free; OVERRIDE;
000335           PROCEDURE TText.FreeSelf(freeParas: BOOLEAN);
000336         {Debugging}
000337           {$IFC fParaTrace}
000338           PROCEDURE  TText.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000339           {$ENDC}
000340
000341          {Calls to textImage procs get routed through these}
000342           PROCEDURE TText.ChangeSelInOtherPanels(textSelection: TTextSelection);
000343           PROCEDURE TText.DelPara(delPara: TEditPara;  fFree: BOOLEAN);
000344           PROCEDURE TText.Draw;
000345           PROCEDURE TText.HiliteRange(highTransit: THighTransit; textRange: TTextRange; wholePara: BOOLEAN);
000346           PROCEDURE TText.HiliteParagraphs(highTransit: THighTransit;
000347                                            startIndex: LONGINT; startLP: INTEGER;
000348                                            endIndex: LONGINT; endLP: INTEGER; wholePara: BOOLEAN);
000349           PROCEDURE TText.InsParaAfter(existingPara: TEditPara;  newPara: TEditPara);
000350           PROCEDURE TText.Invalidate;
000351           PROCEDURE TText.MarkChanged(textRange: TTextRange);
000352           PROCEDURE TText.RecomputeImages;
000353           FUNCTION  TText.SelectAll(textImage: TTextImage): TTextSelection;
000354           END;
000355
000356
000357
000358       TTextImage = SUBCLASS OF TImage
000359           text:               TText;      {complete list of paragraphs}
000360           imageList:          TList;      {paraImages for some range of paragraphs in text}
000361           tickCount:          INTEGER;
000362           growsDynamically:   BOOLEAN;    {TRUE -->  extentLRect bottom grows as more text entered;
000363                                            FALSE ->  text is truncated at last line that fits}
000364           minHeight:          INTEGER;    {the minimum height to shrink (if growsDynamically=TRUE);
000365                                            defaults to height of original extentLRect}
000366
000367           formerBottom:       LONGINT;    {Used by Invalidate when the displayed paragraphs get shorter
000368                                            and text at end needs to be erased}
000369           updateLRect:        LRect;      { "         "                         "              "}
000370
000371           firstLinePixel:     LONGINT;    {Used by Text BB to limit what gets erased on first update line}
000372           useFirstPixel:      BOOLEAN;
000373
000374
000375           { The following fields support multiple linked text images displaying a single text object,
000376             where the text "flows" from one box to the next.  APPLICATIONS ARE RESPONSIBLE FOR
000377             MAINTAINING THESE FIELDS.  This Building Block uses these fields for drawing, etc.
000378             All text images in a chain should have growsDynamically set to FALSE (except possibly
000379             for the last text image in a chain).
```

```
000380            For applications that DO NOT use this feature, the fields will always be as follows:
000381                  startLP = 0;
000382                  endLP =   LP of last character in last paragraph; (if growsDynamically = TRUE)
000383                            LP of last character that fit in extentLRect; (if growsDynamically = FALSE)
000384                  prevTxtImg, nextTxtImg = NIL;
000385                  headTxtImg = SELF;
000386                  tailTxtImg = SELF;
000387           }
000388          firstIndex:     LONGINT;    {index of paragraph at SELF.imageList.First}
000389          startLP:        INTEGER;    {startLP of paragraph at SELF.imageList.First}
000390          endLP:          INTEGER;    {endLP of paragraph at SELF.imageList.Last}
000391
000392          prevTxtImg:     TTextImage; {  for linking textImages that display different parts of }
000393          nextTxtImg:     TTextImage; {  the same text object. eg: columns}
000394          headTxtImg:     TTextImage; {points to first text image in this list}
000395          tailTxtImg:     TTextImage; {points to last text image in this list}
000396
000397        {Creation}
000398          FUNCTION  TTextImage.CREATE(object: TObject; heap: THeap; itsView: TView;
000399                                      itsLRect: LRect; itsText: TText; isGrowable: BOOLEAN): TTextImage;
000400
000401          {TTextImage.Free frees all text images and their paraImages in the text image chain.
000402           It does NOT free any paragraphs, text objects, or paraFormats.  Call this only once
000403           for each text image chain (NOT for each text image in the chain).  Note that TText.Free
000404           frees its textImages so calling this routine is not necessary in most cases}
000405          PROCEDURE  TTextImage.Free; OVERRIDE;
000406
000407          {TTextImage.FreeOneTextImage frees just one text image from the chain.  It pays no attention
000408           to links or whether this is the head text image.  Maintenance of these fields  must be
000409           handled by the caller before calling this routine.  Those who do not use linked text images
000410           should always call TTextImage.Free above, NOT this routine}
000411          PROCEDURE TTextImage.FreeOneTextImage;
000412
000413        {Debugging}
000414          {$IFC fParaTrace}
000415          PROCEDURE TTextImage.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000416          {$ENDC}
000417
000418        {Drawing}
000419          PROCEDURE TTextImage.Draw; OVERRIDE;
000420          PROCEDURE TTextImage.DrawImages(fDraw: BOOLEAN);
000421          PROCEDURE TTextImage.DrawOrInval(invalBits: BOOLEAN);
000422          PROCEDURE TTextImage.HiliteText(highTransit: THighTransit;
000423                                          startIndex: LONGINT; startLP: INTEGER;
000424                                          endIndex: LONGINT; endLP: INTEGER; wholePara: BOOLEAN);
000425
000426        {Locating}
000427          PROCEDURE TTextImage.FindParaAndLp(LPt: LPoint; VAR paraImage: TParaImage;
```

```
000428                                                  VAR paraIndex: LONGINT; VAR aLP: INTEGER);
000429          FUNCTION  TTextImage.FindTextImage(VAR mouseLPt: LPoint; VAR firstTxtImg: TTextImage): TTextImage;
000430          FUNCTION  TTextImage.ImageBottom: LONGINT;
000431          PROCEDURE TTextImage.GetImageRange(firstIndex: LONGINT; VAR firstLP: INTEGER;
000432                                             lastIndex:  LONGINT; VAR lastLP:  INTEGER;
000433                                             VAR firstImage, lastImage: TParaImage);
000434          FUNCTION  TTextImage.ImageWith(paragraph: TEditPara; lp: INTEGER): TParaImage;
000435          PROCEDURE TTextImage.MousePress(mouseLPt: LPoint); OVERRIDE;
000436          PROCEDURE TTextImage.OffsetBy(deltaLPt: LPoint); OVERRIDE;
000437
000438        {Image maintenence}
000439          PROCEDURE TTextImage.AddImage(paraImage: TParaImage);
000440          PROCEDURE TTextImage.DelImagesWith(delPara: TEditPara);
000441          PROCEDURE TTextImage.InsertNewPara(existingPara, newPara: TEditPara);
000442          PROCEDURE TTextImage.InvalAll;
000443          PROCEDURE TTextImage.Invalidate; OVERRIDE; {Invalidate changed lineLRects in changed paraimages}
000444          PROCEDURE TTextImage.MarkChanged(startIndex: LONGINT; startLP: INTEGER;
000445                                           endIndex: LONGINT; endLP: INTEGER);
000446          FUNCTION  TTextImage.NewTextSelection(firstPara: TEditPara; firstIndex: LONGINT; firstLP: INTEGER;
000447                                           lastPara: TEditPara; lastIndex: LONGINT; lastLP: INTEGER
000448                                           ): TTextSelection;
000449          PROCEDURE TTextImage.RecomputeImages(drawAction: TDrawAction; invalBits: BOOLEAN);
000450          PROCEDURE TTextImage.Resize(newExtent: LRect); OVERRIDE;
000451          FUNCTION  TTextImage.SeesSameAs(image: TImage): BOOLEAN; OVERRIDE;
000452
000453        {By default SetFirstIndex just sets firstIndex to  0, but subclassers may override this
000454         if they want the display to start from other than the first paragraph}
000455          PROCEDURE TTextImage.SetFirstIndex;
000456
000457          {These routines are provided so that users can subclass the appropriate class and
000458           then override these methods so that the building block will create the user's subclass
000459           when generating new instances of that class. }
000460          FUNCTION  TTextImage.NewEditPara(initialSize: INTEGER; itsFormat: TParaFormat): TEditPara;
000461          FUNCTION  TTextImage.NewParaImage(itsParagraph: TEditPara; itsLRect: LRect;
000462                                            lineTop: LONGINT; lineLeft: LONGINT): TParaImage;
000463          FUNCTION  TTextImage.NewTextImage(heap: THeap; itsView: TView; itsLRect: LRect;
000464                                            itsText:TText; isGrowable: BOOLEAN): TTextImage;
000465          FUNCTION  TTextImage.TxtImgForClipBoard(heap: THeap; itsView: TView; itsLRect: LRect;
000466                                            itsText:TText; isGrowable: BOOLEAN): TTextImage;
000467          END;
000468
000469
000470      {Clipboard Text View}
000471      TTextView = SUBCLASS OF TView
000472          textImage:  TTextImage;
000473          valid:      BOOLEAN;       {If FALSE, calls Recompute before Drawing}
000474
000475        {Creation}
```

```
000476          FUNCTION  TTextView.CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsExtent: LRect)
000477                                    : TTextView;
000478
000479        {Debugging}
000480          {$IFC fParaTrace}
000481          PROCEDURE TTextView.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000482          {$ENDC}
000483
000484          {$IFC fUseUnivText}
000485          PROCEDURE TTextView.CreateUniversalText; OVERRIDE;
000486          {$ENDC}
000487          PROCEDURE TTextView.Draw; OVERRIDE;
000488          PROCEDURE TTextView.MousePress(mouseLPt: LPoint); OVERRIDE;
000489          END;
000490
000491
000492      {$IFC fUseUnivText}
000493      TTextWriteUnivText = SUBCLASS OF TTKWriteUnivText
000494          textSelection:  TTextSelection;
000495          currIndex:      LONGINT;
000496          currPara:       TEditPara;
000497          currLP:         INTEGER;
000498          currStyleIndex: INTEGER;
000499          currTStyles:     TArray;
000500        {Creation}
000501          FUNCTION  TTextWriteUnivText.CREATE(object: TObject; heap: THeap;
000502                                              itsString: TString; itsDataSize: INTEGER;
000503                                              itsTextSel: TTextSelection): TTextWriteUnivText;
000504
000505        {Debugging}
000506          {$IFC fParaTrace}
000507          PROCEDURE TTextWriteUnivText.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000508          {$ENDC}
000509
000510          PROCEDURE TTextWriteUnivText.FillParagraph; OVERRIDE;
000511          END;
000512      {$ENDC}
000513
000514
000515      TTextSelection = SUBCLASS OF TSelection
000516          textImage:          TTextImage;
000517          textRange:          TTextRange;
000518          isWordSelection:    BOOLEAN;
000519          isParaSelection:    BOOLEAN;
000520          viewTick:           INTEGER;
000521          amTyping:           BOOLEAN;
000522          currTypeStyle:      TTypeStyle;
000523
```

```
000524          FUNCTION  TTextSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
000525                                          itsTextImage: TTextImage; itsAnchorLPt: LPoint;
000526                                          beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
000527                                          endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER
000528                                          ): TTextSelection;
000529
000530
000531        {Debugging}
000532          {$IFC fParaTrace}
000533          PROCEDURE TTextSelection.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000534          {$ENDC}
000535
000536        {Commands}
000537          PROCEDURE TTextSelection.KeyText;
000538          FUNCTION  TTextSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
000539          FUNCTION  TTextSelection.NewStyleCmd(heap: THeap; cmdNumber: TCmdNumber;
000540                                       textImage: TTextImage): TCommand;
000541          FUNCTION  TTextSelection.NewCutCopyCmd(heap: THeap; cmdNumber: TCmdNumber;
000542                                       textImage: TTextImage): TCommand; DEFAULT;
000543          PROCEDURE TTextSelection.StyleFromContext; DEFAULT;
000544          PROCEDURE TTextSelection.DoChangeStyle(cmdNumber: TCmdNumber; paragraph: TParagraph;
000545                                          firstLP: INTEGER; lastLP: INTEGER; VAR newStyle: TTypeStyle);
000546          PROCEDURE TTextSelection.ChangeStyle(cmdNumber: TCmdNumber); DEFAULT;
000547        {Editing}
000548          PROCEDURE TTextSelection.ChangeText(PROCEDURE TextEdit; PROCEDURE Adjust); DEFAULT;
000549          FUNCTION  TTextSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection; DEFAULT;
000550          PROCEDURE TTextSelection.CutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN); DEFAULT;
000551          PROCEDURE TTextSelection.DeleteAndFree; DEFAULT;
000552          FUNCTION  TTextSelection.DeleteButSave: TText; DEFAULT;
000553
000554        {Highlighting}
000555          PROCEDURE TTextSelection.Highlight(highTransit: THighTransit); OVERRIDE;
000556
000557        {Selecting}
000558          FUNCTION  TTextSelection.BecomeInsertionPoint: TInsertionPoint;
000559          PROCEDURE TTextSelection.GetHysteresis(VAR hysterPt: Point); OVERRIDE;
000560          PROCEDURE TTextSelection.MousePress(mouseLPt: LPoint); OVERRIDE;
000561          FUNCTION  TTextSelection.SelSize: INTEGER; ABSTRACT;
000562
000563        {Invalidation}
000564          PROCEDURE TTextSelection.Invalidate; DEFAULT;
000565
000566        {Generate Text Selection in another panel (ie. another Text Image)}
000567          FUNCTION  TTextSelection.ReplicateForOtherPanel(itsTextImage: TTextImage): TTextSelection;
000568          END;
000569
000570
000571      TInsertionPoint = SUBCLASS OF TTextSelection
```

```
000572          typingCmd:          TTypingCmd;     {the current typing command (if user is typing)}
000573          styleCmdNumber:     INTEGER;        {Set to cmdNumber when a type style item is chosen,
000574                                                  set to zero otherwise}
000575          newestLP:           INTEGER;        {the lp position as updated between KeyPause's}
000576          justReturned:       BOOLEAN;        {flag that prevents redundant update in KeyPause}
000577
000578          nextHighTransit:    THighTransit;
000579          nextTransitTime:    LONGINT;
000580
000581        {Creation/Freeing}
000582          FUNCTION  TInsertionPoint.CREATE(object: TObject; heap: THeap; itsView: TView;
000583                                           itsTextImage: TTextImage; itsAnchorLPt: LPoint; itsParagraph: TEditPara;
000584                                           itsIndex: LONGINT; itsLP: INTEGER): TInsertionPoint;
000585
000586        {Debugging}
000587          {$IFC fParaTrace}
000588          PROCEDURE TInsertionPoint.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000589          {$ENDC}
000590
000591        {Commands}
000592          PROCEDURE TInsertionPoint.IdleBegin(centiSeconds: LONGINT); OVERRIDE;
000593          PROCEDURE TInsertionPoint.IdleContinue(centiSeconds: LONGINT); OVERRIDE;
000594          PROCEDURE TInsertionPoint.IdleEnd(centiSeconds: LONGINT); OVERRIDE;
000595          FUNCTION  TInsertionPoint.NewCutCopyCmd(heap: THeap; cmdNumber: TCmdNumber;
000596                                           textImage: TTextImage): TCommand; OVERRIDE;
000597          PROCEDURE TInsertionPoint.StyleFromContext; OVERRIDE;
000598
000599        {Editing}
000600          PROCEDURE TInsertionPoint.CutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN); OVERRIDE;
000601          PROCEDURE TInsertionPoint.FinishPaste(clipSelection: TSelection; pic: PicHandle);
000602          PROCEDURE TInsertionPoint.InsertText(text: TText; isParaSelection: BOOLEAN; isWordSelection: BOOLEAN;
000603                                                                                      universalText: BOOLEAN);
000604          PROCEDURE TInsertionPoint.KeyBack(fWord: BOOLEAN); OVERRIDE;
000605          PROCEDURE TInsertionPoint.KeyChar(ch: CHAR); OVERRIDE;
000606          PROCEDURE TInsertionPoint.KeyClear; OVERRIDE;
000607          PROCEDURE TInsertionPoint.KeyForward(fWord: BOOLEAN); OVERRIDE;
000608
000609        {Selecting}
000610          PROCEDURE TInsertionPoint.MouseMove(mouseLPt: LPoint); OVERRIDE;
000611          PROCEDURE TInsertionPoint.MousePress(mouseLPt: LPoint); OVERRIDE;
000612          PROCEDURE TInsertionPoint.MouseRelease; OVERRIDE;
000613
000614          END;
000615
000616
000617      TOneParaSelection = SUBCLASS OF TTextSelection
000618          anchorBegin:    INTEGER;
000619          anchorEnd:      INTEGER;    {anchorBegin <> anchorEnd iff double or triple click}
```

```
000620
000621        {Creation/Freeing}
000622          FUNCTION  TOneParaSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
000623                                            itsTextImage:  TTextImage; itsAnchorLPt: LPoint; itsParagraph: TEditPara;
000624                                            itsIndex: LONGINT; oldLP: INTEGER; currLP: INTEGER): TOneParaSelection;
000625
000626        {Debugging}
000627          {$IFC fParaTrace}
000628          PROCEDURE TOneParaSelection.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000629          {$ENDC}
000630
000631        {Commands}
000632          PROCEDURE TOneParaSelection.StyleFromContext; OVERRIDE;
000633
000634        {Editing}
000635          FUNCTION  TOneParaSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection; OVERRIDE;
000636          PROCEDURE TOneParaSelection.DeleteAndFree; OVERRIDE;
000637          FUNCTION  TOneParaSelection.DeleteButSave: TText; OVERRIDE;
000638
000639        {Selecting}
000640          PROCEDURE TOneParaSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
000641          PROCEDURE TOneParaSelection.MouseRelease; OVERRIDE;
000642
000643          END;
000644
000645
000646      TMultiParaSelection = SUBCLASS OF TTextSelection
000647          anchorPara:     TEditPara;
000648          anchorIndex:    LONGINT;
000649          anchorBegin:    INTEGER;
000650          anchorEnd:      INTEGER;    {anchorBegin <> anchorEnd iff double or triple click}
000651
000652        {Creation/Freeing}
000653          FUNCTION  TMultiParaSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
000654                                               itsTextImage: TTextImage; itsAnchorLPt: LPoint;
000655                                               beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
000656                                               endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER;
000657                                               beginIsAnchor: BOOLEAN): TMultiParaSelection;
000658
000659        {Debugging}
000660          {$IFC fParaTrace}
000661          PROCEDURE TMultiParaSelection.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000662          {$ENDC}
000663
000664        {Commands}
000665          PROCEDURE TMultiParaSelection.StyleFromContext; OVERRIDE;
000666
000667        {Editing}
```

```
000668          FUNCTION  TMultiParaSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection; OVERRIDE;
000669          FUNCTION  TMultiParaSelection.Delete(saveIt: BOOLEAN): TText;
000670          PROCEDURE TMultiParaSelection.DeleteAndFree; OVERRIDE;
000671          FUNCTION  TMultiParaSelection.DeleteButSave: TText; OVERRIDE;
000672
000673        {Selecting}
000674          PROCEDURE TMultiParaSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
000675          PROCEDURE TMultiParaSelection.MouseRelease; OVERRIDE;
000676
000677          END;
000678
000679
000680                        {-------------  COMMANDS  -----------------}
000681
000682      TClearTextCmd = SUBCLASS OF TCommand
000683
000684        {Variables}
000685          savedText:      TText;      {save the cleared text for undo}
000686          text:           TText;      {the text object we are clearing}
000687
000688        {Creation}
000689          FUNCTION  {TClearTextCmd.}CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
000690                                           itsImage: TImage; itsText: TText): TClearTextCmd;
000691
000692          PROCEDURE TClearTextCmd.Free; OVERRIDE;
000693          {$IFC fParaTrace}
000694          PROCEDURE TClearTextCmd.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000695          {$ENDC}
000696        {Command Execution}
000697          PROCEDURE TClearTextCmd.Commit; OVERRIDE;
000698          PROCEDURE TClearTextCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
000699          END;
000700
000701      TStyleCmd = SUBCLASS OF TCommand
000702
000703        {Variables}
000704          text:               TText;
000705          textSelection:      TTextSelection;
000706          firstFiltParaIndex: LONGINT;
000707          lastFiltParaIndex:  LONGINT;
000708          filtFirstLP:        INTEGER;
000709          filtLastLP:         INTEGER;
000710          currFilteredPara:   TEditPara;      {handle to most recently filtered paragraph}
000711          filteredStyles:     TArray;         {changed type styles of most recently filtered paragraph}
000712
000713        {Creation}
000714          FUNCTION  TStyleCmd.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
000715                                     itsImage: TImage;
```

```
000716                                  itsFirstIndex: LONGINT; itsLastIndex:  LONGINT;
000717                                  itsLPFirst: INTEGER; itsLPLast: INTEGER;
000718                                  itsSelection: TTextSelection): TStyleCmd;
000719
000720          PROCEDURE TStyleCmd.Free; OVERRIDE;
000721          {$IFC fParaTrace}
000722          PROCEDURE TStyleCmd.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000723          {$ENDC}
000724        {Command Execution}
000725          PROCEDURE TStyleCmd.Commit; OVERRIDE;
000726          PROCEDURE TStyleCmd.FilterAndDo(actualObject: TObject;
000727                                          PROCEDURE DoToObject(filteredObject: TObject)); OVERRIDE;
000728          PROCEDURE TStyleCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
000729          END;
000730
000731      TTextCutCopy = SUBCLASS OF TCutCopyCommand
000732
000733        {Variables}
000734          text:   TText;
000735
000736        {Creation}
000737          FUNCTION  TTextCutCopy.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
000738                                        itsImage: TImage;
000739                                       isCutCmd: BOOLEAN; itsText: TText): TTextCutCopy;
000740
000741          PROCEDURE TTextCutCopy.Free; OVERRIDE;
000742          {$IFC fParaTrace}
000743          PROCEDURE TTextCutCopy.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000744          {$ENDC}
000745        {Command Execution}
000746          PROCEDURE TTextCutCopy.DoCutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN;
000747                                           cmdPhase: TCmdPhase); OVERRIDE;
000748          END;
000749
000750      TTextPaste = SUBCLASS OF TPasteCommand
000751
000752        {Variables}
000753          savedText:  TText;
000754          pasteRange: TTextRange;     {The text range spanned by the pasted text}
000755          text:       TText;
000756          origIsPara: BOOLEAN;
000757          origIsWord: BOOLEAN;
000758          clipIsPara: BOOLEAN;
000759
000760        {Creation}
000761          FUNCTION  TTextPaste.CREATE(object: TObject; heap: THeap;  itsImage: TImage;
000762                                      itsText: TText): TTextPaste;
000763
```

```
000764          PROCEDURE TTextPaste.Free; OVERRIDE;
000765          {$IFC fParaTrace}
000766          PROCEDURE TTextPaste.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000767          {$ENDC}
000768        {Command Execution}
000769          PROCEDURE TTextPaste.Commit; OVERRIDE;
000770          PROCEDURE TTextPaste.DoPaste(clipSelection: TSelection; pic: PicHandle; cmdPhase: TCmdPhase);
000771                                                                                          OVERRIDE;
000772          END;
000773
000774
000775      TTypingCmd = SUBCLASS OF TCommand
000776
000777        {Variables}
000778          savedText:          TText;
000779          text:               TText;
000780          newCharCount:       INTEGER;
000781          newParaCount:       INTEGER;
000782          typingRange:        TTextRange;     {The text range spanned by the typed characters}
000783          otherInsPts:        TList;
000784
000785        {Creation}
000786          FUNCTION  TTypingCmd.CREATE(object: TObject; heap: THeap; itsImage: TImage;
000787                                      itsText: TText): TTypingCmd;
000788
000789          PROCEDURE TTypingCmd.Free; OVERRIDE;
000790          {$IFC fParaTrace}
000791          PROCEDURE TTypingCmd.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
000792          {$ENDC}
000793        {Command Execution}
000794          PROCEDURE TTypingCmd.Commit; OVERRIDE;
000795          PROCEDURE TTypingCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
000796          END;
000797
000798
000799  VAR fParaTrace:             BOOLEAN;
000800      fTextTrace:             BOOLEAN;
000801
000802
000803  IMPLEMENTATION
000804
000805  (*
000806  {$I UTEXT2.text}  {Paragraph classes}
000807  {$I UTEXT3.text}  {TStyleSheet, TText, TTextImage, TTextView}
000808  {$I UTEXT4.text}  {Text Selections and Commands}
000809  *)
000810
000811  {$I LibTK/UTEXT2.text}  {Paragraph classes}
```

```
000812  {$I LibTK/UTEXT3.text}  {TStyleSheet, TText, TTextImage, TTextView}
000813  {$I LibTK/UTEXT4.text}  {Text Selections and Commands}
000814  END.
000815

End of File -- Lines: 815 Characters: 35994
```

```
================================================================================================
FILE: "LIBTK/UTEXT2.TEXT"
================================================================================================

000001  {UText2}    {Paragraph Classes}
000002
000003  {changed 05/11/84 1135  Added TParagraph.Clone}
000004  {changed 04/25/84 1250  Changed FilterAndDo calls back to filtering TParaImage for Compugraphic}
000005  {changed 04/18/84 1652  Use TTextImage.firstLinePixel in DrawParaImage}
000006  {changed 04/17/84 1806  Put call to ReplTString outside of IF in ReplPara;
000007                          Put more parameter checks in ReplTString}
000008  {changed 04/16/84 1135  Added styleSheet field to TParaFormat; use it in ChangeRefCountBy}
000009  {changed 04/16/84 1033  Put PicTextBegin, End in TParagraph.DrawLine;
000010                          Put PicGrpBegin, End in TParaImage.RedrawLines;
000011                          Removed picture comments from TParaImage.DrawLine}
000012  {changed 04/13/84 1739  Set paraformat.refcount = 0 in TParaFormat.Clone}
000013  {changed 04/13/84 1537  Changed calls to FilterAndDo to pass TEditPara rather than TParaImage}
000014  {changed 04/12/84 2344  Modified UpdateRuns to use new parameter list}
000015  {changed 04/11/84 1527  Call UpdateRuns after deleting characters in TParagraph.ReplPString and ReplTString}
000016  {changed 04/11/84 1454  Debug statement in Qualifies to check bug involving special characters}
000017  {changed 04/10/84 1400  Changed TEditPara.images field back to a TList and adusted references to it}
000018  {changed 04/10/84 1158  Put calls to TParaFormat.ChangeRefCountBy in TEditPara.CREATE, Free}
000019
000020  {$IFC fRngABC}
000021  {$R+}
000022  {$ELSEC}
000023  {$R-}
000024  {$ENDC}
000025
000026  {$IFC fSymABC}
000027  {$D+}
000028  {$ELSEC}
000029  {$D-}
000030  {$ENDC}
000031
000032  TYPE
000033      TScanState = (cBeforeRange, cInRange, cAfterRange);
000034      TFakeTStyle = PACKED ARRAY[1..SIZEOF(TTypeStyle)] OF CHAR;
000035      {$IFC LibraryVersion <= 20}
000036      Style = TSeteface;
000037      FontInfo = TFinfo;
000038      {$ENDC}
000039
000040
000041  {$S SgTxtHot}
000042
000043  VAR nextHighTransit:    THighTransit;
```

```
000044      nextTransitTime:    LONGINT;
000045      uvFont:             ARRAY [1..14] OF TFontRecord;
000046
000047
000048  METHODS OF TParaFormat;
000049
000050  {$S SgTxtIni}
000051      FUNCTION TParaFormat.CREATE(object: TObject; heap: THeap; itsStyleSheet: TStyleSheet): TParaFormat;
000052      VAR tabArray:   TArray;
000053      BEGIN
000054          {$IFC fTrace}BP(6);{$ENDC}
000055          IF object = NIL THEN
000056              object := NewObject(heap, THISCLASS);
000057          SELF := TParaFormat(object);
000058          tabArray := TArray.CREATE(NIL, heap, 0, SIZEOF(TTxtTabDescriptor));
000059          WITH SELF DO
000060              BEGIN
000061              {$H-}
000062              MakeTypeStyle(famModern, size12Pitch, [], dfltTstyle);
000063              {$H+}
000064              wordWrap := TRUE;
000065              quad := aLeft;
000066              firstIndent := 0;
000067              leftIndent := 0;
000068              rightIndent := 0;
000069              spaceAbovePara := 0;
000070              spaceBelowPara := 0;
000071              lineSpacing := 0;
000072              tabs := tabArray;
000073              permanent := FALSE;
000074              refCount := 0;
000075              styleSheet := itsStyleSheet;
000076              END;
000077          {$IFC fTrace}EP;{$ENDC}
000078      END;
000079
000080  {$S SgTxtCld}
000081      PROCEDURE TParaFormat.Free;
000082      BEGIN
000083          {$IFC fTrace}BP(10);{$ENDC}
000084          Free(SELF.tabs);
000085          SUPERSELF.Free;
000086          {$IFC fTrace}EP;{$ENDC}
000087      END;
000088
000089
000090  {$S SgTxtCld}
000091      FUNCTION TParaFormat.Clone(heap: THeap): TObject;
```

```
000092      VAR tabs:       TArray;
000093          paraFormat: TParaFormat;
000094      BEGIN
000095          {$IFC fTrace}BP(10);{$ENDC}
000096          tabs := TArray(SELF.tabs.Clone(heap));
000097          paraFormat := TParaFormat(SUPERSELF.Clone(heap));
000098          paraFormat.tabs := tabs;
000099          paraFormat.refCount := 0;
000100          Clone := paraFormat;
000101          {$IFC fTrace}EP;{$ENDC}
000102      END;
000103
000104
000105  {$S SgTxtCld}
000106  {$IFC fParaTrace}
000107      PROCEDURE TParaFormat.Fields(PROCEDURE Field(nameAndType: S255));
000108      BEGIN
000109          SUPERSELF.Fields(Field);
000110          Field('dfltTstyle: RECORD onFaces: HexByte; filler: HexByte; fontFamily: Byte; fontSize: Byte END');
000111          Field('wordWrap: BOOLEAN');
000112          Field('quad: HexByte');
000113          Field('firstIndent: INTEGER');
000114          Field('leftIndent: INTEGER');
000115          Field('rightIndent: INTEGER');
000116          Field('spaceAbovePara: INTEGER');
000117          Field('spaceBelowPara: INTEGER');
000118          Field('lineSpacing: INTEGER');
000119          Field('tabs: TArray');
000120          Field('refCount: INTEGER');
000121          Field('permanent: BOOLEAN');
000122          Field('styleSheet: BOOLEAN');
000123          Field('');
000124      END;
000125  {$ENDC}
000126
000127
000128  {$S SgTxtCld}
000129      PROCEDURE TParaFormat.ChangeRefCountBy(delta: INTEGER);
000130      BEGIN
000131          {$IFC fTrace}BP(10);{$ENDC}
000132          SELF.refCount := SELF.refCount + delta;
000133          IF (SELF.refCount <= 0) AND NOT SELF.permanent THEN
000134              BEGIN
000135              IF SELF.styleSheet <> NIL THEN
000136                  SELF.styleSheet.formats.DelObject(SELF, TRUE)
000137              ELSE
000138                  SELF.Free;
000139              END;
```

```
000140          {$IFC fTrace}EP;{$ENDC}
000141      END;
000142
000143
000144  {$S SgTxtHot}
000145      PROCEDURE TParaFormat.SetTypeStyle(tStyle: TTypeStyle);
000146      BEGIN
000147          {$IFC fTrace}BP(10);{$ENDC}
000148          SetQDTypeStyle(tStyle);
000149          {$IFC fTrace}EP;{$ENDC}
000150      END;
000151
000152
000153  {$S SgTxtIni}
000154  BEGIN
000155      UnitAuthor('Apple');
000156  END;
000157
000158
000159  METHODS OF TParagraph;
000160
000161  {$S SgTxtIni}
000162      FUNCTION TParagraph.CREATE(object: TObject; heap: THeap;
000163                                          initialSize: INTEGER; initialTypeStyle: TTypeStyle): TParagraph;
000164      VAR ts:             TArray;
000165          styleChange:    TStyleChange;
000166      BEGIN
000167          {$IFC fTrace}BP(10);{$ENDC}
000168          IF object = NIL THEN
000169              object := NewDynObject(heap, THISCLASS, initialSize);
000170          SELF := TParagraph(TString.CREATE(object, heap, initialSize));
000171
000172          ts := TArray.CREATE(NIL, heap, 0, SIZEOF(TStyleChange));
000173
000174          styleChange.lp := MAXINT; { -sentinel- }
000175          styleChange.newStyle := initialTypeStyle;
000176          ts.InsAt(1, @styleChange);
000177          styleChange.lp := -1;
000178          ts.InsAt(1, @styleChange);
000179
000180          SELF.typeStyles := ts;
000181          {$IFC fTrace}EP;{$ENDC}
000182      END;
000183
000184  {$S SgTxtCld}
000185      PROCEDURE TParagraph.Free;
000186      BEGIN
000187          {$IFC fTrace}BP(10);{$ENDC}
```

```
000188          Free(SELF.typeStyles);
000189          SUPERSELF.Free;
000190          {$IFC fTrace}EP;{$ENDC}
000191      END;
000192
000193
000194  {$S SgTxtCld}
000195      FUNCTION TParagraph.Clone(heap: THeap): TObject;
000196      VAR paragraph:  TParagraph;
000197          styles:     TArray;
000198      BEGIN
000199          {$IFC fTrace}BP(10);{$ENDC}
000200          styles := TArray(SELF.typeStyles.Clone(heap));
000201          paragraph := TParagraph(SUPERSELF.Clone(heap));
000202          paragraph.typeStyles := styles;
000203          Clone := paragraph;
000204          {$IFC fTrace}EP;{$ENDC}
000205      END;
000206
000207
000208  {$S SgTxtCld}
000209  {$IFC fParaTrace}
000210      PROCEDURE TParagraph.Fields(PROCEDURE Field(nameAndType: S255));
000211      BEGIN
000212          SUPERSELF.Fields(Field);
000213          Field(CONCAT('typeStyles: TArray OF RECORD lp: INTEGER; onFaces: HexByte; ',
000214                                                  'filler: HexByte; fontFamily: Byte; fontSize: Byte END'));
000215          Field('');
000216      END;
000217  {$ENDC}
000218
000219
000220  {$S TK2Start}
000221    {BuildExtentLRect takes an LPoint that indicates the baseline of the paragraph.  It returns
000222     in extentLRect the bounding rectangle whose height is based on the tallest font in the
000223     paragraph and width is the width of the characters in the paragraph.}
000224      PROCEDURE TParagraph.BuildExtentLRect(baseLPt: LPoint; VAR extentLRect: LRect);
000225      VAR styleChange:    TStyleChange;
000226          fInfo:          FontInfo;
000227          i:              INTEGER;
000228          tallestFont:    FontInfo;
000229          width:          INTEGER;
000230          oldTallest:     INTEGER;
000231      BEGIN
000232          {$IFC fTrace}BP(10);{$ENDC}
000233          oldTallest := 0;
000234          FOR i := 1 to SELF.typeStyles.size - 1 DO
000235              BEGIN
```

```
000236              SELF.typeStyles.GetAt(i, @styleChange);
000237              SELF.SetTypeStyle(styleChange.newStyle);
000238              GetFontInfo(fInfo);
000239              WITH fInfo DO
000240                  IF oldTallest < ascent + descent + leading THEN
000241                      BEGIN
000242                      oldTallest := ascent + descent + leading;
000243                      tallestFont := fInfo;
000244                      END;
000245              END;
000246          width := SELF.Width(1, SELF.size);
000247          WITH extentLRect, baseLPt, tallestFont DO
000248              BEGIN
000249              top := v - ascent;
000250              bottom := v + descent + leading;
000251              left := h;
000252              right := h + width;
000253              END;
000254          {$IFC fTrace}EP;{$ENDC}
000255      END;
000256
000257
000258  {$S SgTxtCld}
000259      PROCEDURE TParagraph.ChangeStyle(startLP, endLP: INTEGER; PROCEDURE Change(VAR typeStyle: TTypeStyle);
000260                                                                VAR styleOfStartLP: TTypeStyle);
000261      VAR firstChange:    TStyleChange;
000262          tempChange:     TStyleChange;
000263          prevChange:     TStyleChange;
000264          styles:         TArray;
000265          styleIndex:     INTEGER;
000266          newStyle:       TTypeStyle;
000267      BEGIN
000268          {$IFC fTrace}BP(10);{$ENDC}
000269          {$IFC fParaTrace}
000270          IF fParaTrace THEN
000271              BEGIN
000272              WriteLn('=== Entering TParagraph.ChangeStyle: startLP=', startLP:1, '  endLP=', endLP:1);
000273              END;
000274          {$ENDC}
000275          styles := SELF.typeStyles;
000276
000277          styleIndex := 1;
000278          REPEAT
000279              styleIndex := styleIndex + 1;
000280              styles.GetAt(styleIndex, @tempChange);
000281          UNTIL tempChange.lp >= startLP;
000282
000283          {If the change is on a run boundary, just remember the changed style at the beginning so
```

```
000284          we can set styleOfStartLP later}
000285         IF tempChange.lp = startLP THEN
000286             BEGIN
000287             firstChange := tempChange;
000288             Change(firstChange.newStyle);
000289             END
000290         ELSE
000291             BEGIN
000292             {Insert new run descriptor for beginning of changed characters}
000293             styles.GetAt(styleIndex - 1, @firstChange);
000294             prevChange := firstChange;
000295             firstChange.lp := startLP;
000296             Change(firstChange.newStyle);
000297             styles.InsAt(styleIndex, @firstChange);
000298             styleIndex := styleIndex + 1;
000299             END;
000300
000301         {Change existing runs}
000302         WHILE (tempChange.lp < endLP) DO
000303             BEGIN
000304             prevChange := tempChange;
000305             Change(tempChange.newStyle);
000306             styles.PutAt(styleIndex, @tempChange);
000307             styleIndex := styleIndex + 1;
000308             styles.GetAt(styleIndex, @tempChange);
000309             END;
000310
000311         {Don't restore old run info if new run info goes to end of para or ends on old run boundary}
000312         IF endLP < SELF.size THEN
000313             IF tempChange.lp <> endLP THEN
000314                 BEGIN
000315                 prevChange.lp := endLP;
000316                 styles.InsAt(styleIndex, @prevChange);
000317                 END;
000318
000319         SELF.CleanRuns;
000320
000321         {return typestyle of beginning of run}
000322         styleOfStartLP := firstChange.newStyle;
000323         {$IFC fTrace}EP;{$ENDC}
000324     END;
000325
000326
000327 {$S SgTxtCld}
000328     PROCEDURE TParagraph.ChgFace(startLP, endLP: INTEGER;
000329                                  newOnFaces: {$IFC LibraryVersion <= 20}TSeteface{$ELSEC}Style{$ENDC};
000330                                                  VAR styleOfStartLP: TTypeStyle);
000331         PROCEDURE ChangeFace(VAR typeStyle: TTypeStyle);
```

```
000332          BEGIN
000333              IF newOnFaces = [] THEN
000334                  typeStyle.onFaces := []
000335              ELSE
000336                  typeStyle.onFaces := typeStyle.onFaces + newOnFaces;
000337          END;
000338      BEGIN
000339          {$IFC fTrace}BP(10);{$ENDC}
000340          SELF.ChangeStyle(startLP, endLP, ChangeFace, styleOfStartLP);
000341          {$IFC fTrace}EP;{$ENDC}
000342      END;
000343
000344
000345  {$S SgTxtCld}
000346      PROCEDURE TParagraph.ChgFontFamily(startLP, endLP: INTEGER; newFontFamily: Byte;
000347                                                          VAR styleOfStartLP: TTypeStyle);
000348          PROCEDURE ChangeFamily(VAR typeStyle: TTypeStyle);
000349          BEGIN
000350              typeStyle.font.fontFamily := newFontFamily;
000351          END;
000352      BEGIN
000353          {$IFC fTrace}BP(10);{$ENDC}
000354          SELF.ChangeStyle(startLP, endLP, ChangeFamily, styleOfStartLP);
000355          {$IFC fTrace}EP;{$ENDC}
000356      END;
000357
000358
000359  {$S SgTxtCld}
000360      PROCEDURE TParagraph.ChgFontSize(startLP, endLP: INTEGER; newFontSize: Byte;
000361                                                          VAR styleOfStartLP: TTypeStyle);
000362          PROCEDURE ChangeSize(VAR typeStyle: TTypeStyle);
000363          BEGIN
000364              typeStyle.font.fontSize := newFontSize;
000365          END;
000366      BEGIN
000367          {$IFC fTrace}BP(10);{$ENDC}
000368          SELF.ChangeStyle(startLP, endLP, ChangeSize, styleOfStartLP);
000369          {$IFC fTrace}EP;{$ENDC}
000370      END;
000371
000372
000373  {$S SgTxtCld}
000374      {Deletes redundant run information}
000375      PROCEDURE TParagraph.CleanRuns;
000376      VAR styleChange:    TStyleChange;
000377          prevChange:     TStyleChange;
000378          styles:         TArray;
000379          styleIndex:     INTEGER;
```

```
000380      BEGIN
000381          {$IFC fTrace}BP(10);{$ENDC}
000382          styles := SELF.typeStyles;
000383          styles.GetAt(1, @prevChange);
000384          styleIndex := 2;
000385          {Iterate through the style changes and delete any that have either the same lp as the previous
000386           change or the same font and faces info}
000387          WHILE styleIndex < styles.Size DO
000388              BEGIN
000389              styles.GetAt(styleIndex, @styleChange);
000390              IF (styleChange.lp = prevChange.lp) OR
000391                          ((styleChange.newStyle.onFaces = prevChange.newStyle.onFaces) AND
000392                          (styleChange.newStyle.font.fontNum = prevChange.newStyle.font.fontNum)) THEN
000393                  styles.DelAt(styleIndex)
000394              ELSE
000395                  styleIndex := styleIndex + 1;
000396              prevChange := styleChange;
000397              END;
000398          {$IFC fTrace}EP;{$ENDC}
000399      END;
000400
000401  {$S SgTxtHot}
000402      PROCEDURE TParagraph.Draw(i: LONGINT; howMany: INTEGER);
000403      VAR dumInt:     INTEGER;
000404          dumIndex:   INTEGER;
000405      BEGIN
000406          {$IFC fTrace}BP(10);{$ENDC}
000407          dumIndex := 1;
000408          SELF.DrawLine(i-1, i + howMany - 2, TRUE, FALSE, dumInt, dumIndex);
000409          {$IFC fTrace}EP;{$ENDC}
000410      END;
000411
000412
000413  {$S TK2Start}
000414      PROCEDURE TParagraph.DrawLine(startLP, endLP: INTEGER; fDraw: BOOLEAN; fWidth: BOOLEAN;
000415                                          VAR width: INTEGER; VAR styleIndex: INTEGER);
000416      {If fDraw = TRUE, draws a line of characters from startLP to endLP; does not worry about word wrap.
000417       If fWidth = TRUE, returns width of characters.  Also accepts an initial styleIndex (index into
000418       run array) to make typestyle scanning faster. Returns styleIndex of run of last character drawn.}
000419
000420      {IDEAS TO MAKE THIS FASTER:
000421          special check to see if there are no typestyle changes in this para?
000422
000423      }
000424
000425
000426      VAR startPP:        INTEGER;
000427          endPP:          INTEGER;
```

```
000428          styleChange:    TStyleChange;
000429          prevChange:     TStyleChange;
000430          tStyles:        TArray;
000431          drawCount:      INTEGER;
000432      BEGIN
000433          {$IFC fTrace}BP(10);{$ENDC}
000434          {$IFC fParaTrace}
000435          IF fParaTrace THEN
000436              BEGIN
000437              writeln('>> DrawLine: startLP=', startLP:1, '  endLP=', endLP:1,
000438                      '  styleIndex=', styleIndex:1);
000439              WriteLn('>> DrawLine: fDraw=', fDraw, '  holeStart =', SELF.holeStart:1,
000440                      '  holeSize =', SELF.holeSize:1);
000441              END;
000442          {$ENDC}
000443          width := 0;
000444
000445          tStyles := SELF.typeStyles;
000446          IF (styleIndex < 1) OR (styleIndex > tStyles.size) THEN
000447              styleIndex := 1;
000448
000449          tStyles.GetAt(styleIndex, @styleChange);
000450          prevChange := styleChange;
000451          WHILE styleChange.lp < startLP DO
000452              BEGIN
000453              prevChange := styleChange;
000454              styleIndex := styleIndex + 1;
000455              tStyles.GetAt(styleIndex, @styleChange);
000456              END;
000457
000458          PicTextBegin(aLeft);
000459          SELF.SetTypeStyle(prevChange.newStyle);
000460
000461          {$IFC fParaTrace}
000462          IF fParaTrace THEN
000463              writeln('>> DrawLine: starting to Draw');
000464          {$ENDC}
000465          WHILE startLP <= endLP DO
000466              BEGIN
000467              drawCount := MIN(styleChange.lp, endLP+1) - startLP;
000468              IF fWidth THEN
000469                  width := width + TString.Width(startLP+1, drawCount);
000470              IF fDraw THEN
000471                  BEGIN
000472                  {$IFC fParaTrace}
000473                  IF fParaTrace THEN
000474                      writeln('>> DrawLine: About to call DrawText; startLP,drawCount=', startLP:1,
000475                                      ',', drawCount:1);
```

```
000476                  {$ENDC}
000477                  TString.Draw(startLP+1, drawCount);
000478                  END;
000479              startLP := startLP + drawCount;
000480              IF startLP = styleChange.lp THEN
000481                  BEGIN
000482                  {$IFC fParaTrace}
000483                  IF fParaTrace THEN
000484                      writeln('>> DrawLine: found a typestyle change at LP=', startLP:1);
000485                  {$ENDC}
000486                  SELF.SetTypeStyle(styleChange.newStyle);
000487                  styleIndex := styleIndex+1;
000488                  tStyles.GetAt(styleIndex, @styleChange)
000489                  END;
000490              END;
000491
000492          PicTextEnd;
000493          styleIndex := styleIndex-1; {return styleIndex of current typeStyle run}
000494          {$IFC fParaTrace}
000495          IF fParaTrace THEN
000496              BEGIN
000497              Writeln('>> DrawLine: Finished, width=', width:1);
000498              WriteLn;
000499              END;
000500          {$ENDC}
000501          {$IFC fTrace}EP;{$ENDC}
000502      END;
000503
000504
000505  {$S SgTxtWrm}
000506      PROCEDURE TParagraph.FindWordBounds(orig: INTEGER; VAR first, last: INTEGER);
000507      BEGIN
000508          {$IFC fTrace}BP(10);{$ENDC}
000509          first := orig;
000510          last := orig;
000511          IF SELF.Qualifies(orig) THEN
000512              BEGIN
000513              WHILE SELF.Qualifies(first - 1) DO first := first - 1;
000514              WHILE SELF.Qualifies(last + 1) DO last := last + 1;
000515              END;
000516          IF last < SELF.size THEN
000517              last := last+1; {always selects at least one character, except at end of para}
000518          {$IFC fTrace}EP;{$ENDC}
000519      END;
000520
000521
000522  {$S SgTxtHot}
000523      FUNCTION TParagraph.FixLP(LP: INTEGER): INTEGER;
```

```
000524      BEGIN
000525          {$IFC fTrace}BP(6);{$ENDC}
000526          IF LP < 0 THEN
000527              FixLP := 0
000528          ELSE IF LP >= SELF.size THEN
000529              FixLP := SELF.size
000530          ELSE
000531              FixLP := LP;
000532          {$IFC fTrace}EP;{$ENDC}
000533      END;
000534
000535
000536  {$S SgTxtCld}
000537      PROCEDURE TParagraph.NewStyle(startLP, endLP: INTEGER; newTypeStyle: TTypeStyle);
000538      VAR styleOfStartLP: TTypeStyle;
000539          PROCEDURE ChgStyle(VAR typeStyle: TTypeStyle);
000540          BEGIN
000541              typeStyle := newTypeStyle;
000542          END;
000543      BEGIN
000544          {$IFC fTrace}BP(10);{$ENDC}
000545          SELF.ChangeStyle(startLP, endLP, ChgStyle, styleOfStartLP);
000546          {$IFC fTrace}EP;{$ENDC}
000547      END;
000548
000549
000550  {$S SgTxtWrm}
000551      FUNCTION TParagraph.Qualifies(pos: INTEGER): BOOLEAN;
000552      VAR i,j:    INTEGER;
000553          left, this, right:  CHAR;
000554
000555          FUNCTION CharClass(ch: CHAR): CHAR;
000556          VAR c:  INTEGER;
000557          BEGIN
000558              c := ORD(ch);
000559              IF c IN [65..90, 97..122, 128..159, 167, 174..175,
000560                       187..188, 190..191, 202] THEN ch := 'A'
000561              ELSE IF (48 <= c) AND (c <= 57) THEN ch := '9'
000562              ELSE IF (c = 162) OR (c = 163) OR (c = 180) THEN ch := '$';
000563              CharClass := ch;
000564          END;
000565
000566          FUNCTION CharAt(i: INTEGER): CHAR;
000567          BEGIN
000568              IF i < 0 THEN
000569                  CharAt := ' '
000570              ELSE IF i >= SELF.size THEN
000571                  CharAt := ' '
```

```
000572              ELSE
000573                  CharAt := SELF.At(i+1);
000574          END;
000575
000576      BEGIN {Qualifies}
000577          {$IFC fTrace}BP(9);{$ENDC}
000578          left := CharClass(CharAt(pos-1));
000579          this := CharClass(CharAt(pos));
000580          right := CharClass(CharAt(pos+1));
000581          {$IFC fParaTrace}
000582          IF fParaTrace THEN
000583              WriteLn('IN QUALIFIES: left, this, right = (', left:1, this:1, right:1, ')  => [',
000584                          ORD(left):1, ',', ORD(this):1, ',', ORD(right):1, ']');
000585          {$ENDC}
000586
000587          FOR i := 1 TO (LENGTH(wordDelimiters) + 1) DIV 4 DO
000588              BEGIN
000589              j := 4*i-2; { FOR j := 2 TO LENGTH(wordDelimiters) STEP 4 DO }
000590              IF ((wordDelimiters[j-1]=left) OR (wordDelimiters[j-1] = 'x')) AND
000591                  (wordDelimiters[j] = this) AND
000592                 ((wordDelimiters[j+1]=right) OR (wordDelimiters[j+1]='x')) THEN
000593                      BEGIN
000594                      Qualifies := TRUE;
000595                      {$IFC fTrace}EP;{$ENDC}
000596                      EXIT(Qualifies);
000597                      END;
000598              END;
000599          Qualifies := FALSE;
000600          {$IFC fTrace}EP;{$ENDC}
000601      END;
000602
000603
000604  {$S SgTxtWrm}
000605      PROCEDURE TParagraph.ReplPara(fPos, numChars: INTEGER;
000606                                          otherPara: TParagraph; otherFPos, otherNumChars: INTEGER);
000607      VAR styles:         TArray;
000608          otherStyles:    TArray;
000609          styleIndex:     INTEGER;
000610          otherIndex:     INTEGER;
000611          styleChange:    TStyleChange;
000612          otherChange:    TStyleChange;
000613          prevStyle:      TTypeStyle;
000614          diff:           INTEGER;
000615          endLP:          INTEGER;
000616          nextLP:         INTEGER;
000617      BEGIN
000618          {$IFC fTrace}BP(9);{$ENDC}
000619          otherPara.StopEdit;
```

```
000620          SELF.ReplTString(fPos, numChars, otherPara, otherFPos, otherNumChars);
000621          IF otherNumChars > 0 THEN
000622              BEGIN
000623
000624              {COPY THE TYPESTYLE RUN INFO TO SELF.typeStyles}
000625              otherStyles := otherPara.typeStyles;
000626              styles := SELF.typeStyles;
000627              {Find out what run we're in in SELF}
000628              styleIndex := 1;
000629              REPEAT
000630                  styleIndex := styleIndex+1;
000631                  styles.GetAt(styleIndex, @styleChange);
000632              UNTIL fPos <= styleChange.lp;
000633              nextLP := styleChange.lp;
000634              IF fPos < nextLP THEN
000635                  styles.GetAt(styleIndex-1, @styleChange); {back up one to get current run}
000636
000637              {Find the first run in otherPara}
000638              otherIndex := 1;
000639              REPEAT
000640                  otherIndex := otherIndex+1;
000641                  otherStyles.GetAt(otherIndex, @otherChange);
000642              UNTIL otherFPos < otherChange.lp;
000643              otherStyles.GetAt(otherIndex-1, @otherChange);
000644
000645              diff := fPos - otherFPos;
000646              endLP := otherfPos + otherNumChars;
000647              prevStyle := styleChange.newStyle;
000648              {Insert the new run info but avoid consecutive run descriptors of the same info}
000649              otherChange.lp := otherFPos;
000650              WHILE otherChange.lp <= endLP DO
000651                  BEGIN
000652                  IF TFakeTStyle(otherChange.newStyle) <> TFakeTStyle(prevStyle) THEN
000653                      BEGIN
000654                      otherChange.lp := otherChange.lp + diff;
000655                      styles.InsAt(styleIndex, @otherChange);
000656                      styleIndex := styleIndex + 1;
000657                      prevStyle := otherChange.newStyle;
000658                      END;
000659                  otherStyles.GetAt(otherIndex, @otherChange);
000660                  otherIndex := otherIndex + 1;
000661                  END;
000662
000663              {Insert descriptor of original run after the inserted info, unless we were on a run boundary
000664                or the last run has the same font and faces as the original run}
000665              IF fPos < nextLP THEN
000666                  IF TFakeTStyle(styleChange.newStyle) <> TFakeTStyle(prevStyle) THEN
000667                      BEGIN
```

```
000668                      styleChange.lp := fPos+otherNumChars;
000669                      styles.InsAt(styleIndex, @styleChange);
000670                      END;
000671                  END;
000672              {$IFC fTrace}EP;{$ENDC}
000673          END;
000674
000675
000676      {$S SgTxtHot}
000677          PROCEDURE TParagraph.ReplPString(fPos, numChars: INTEGER; pStr: TPString);
000678              VAR otherNumChars: INTEGER;
000679          BEGIN
000680              {$IFC fTrace}BP(9);{$ENDC}
000681
000682              { make fPos lie within [0..# chars in paragraph] }
000683              fPos := SELF.fixLP(fPos);
000684
000685              IF pStr = NIL THEN
000686                  otherNumChars := 0
000687              ELSE
000688                  otherNumChars := Length(pStr^);
000689
000690              SELF.StartEdit(otherNumChars);
000691              SELF.DelManyAt(fPos + 1, numChars);
000692              (*
000693              SELF.UpdateRuns(fPos,-numChars);
000694              *)
000695              IF pStr <> NIL THEN
000696                  SELF.InsPStrAt(fPos + 1, pStr);
000697              SELF.StopEdit;
000698
000699              SELF.UpdateRuns(fPos, numChars, otherNumChars);
000700              {$IFC fTrace}EP;{$ENDC}
000701          END;
000702
000703
000704      {$S SgTxtHot}
000705          PROCEDURE TParagraph.ReplTString(fPos, numChars: INTEGER;
000706                              otherString: TString; otherFPos, otherNumChars: INTEGER);
000707          BEGIN
000708              {$IFC fTrace}BP(9);{$ENDC}
000709
000710              { make fPos lie within [0..# chars in paragraph] }
000711              fPos := SELF.fixLP(fPos);
000712
000713              SELF.StartEdit(otherNumChars);
000714
000715              IF numChars > 0 THEN
```

```
000716              SELF.DelManyAt(fPos + 1, numChars);
000717
000718          IF (otherString <> NIL) AND (otherNumChars > 0) THEN
000719              SELF.InsManyAt(fPos + 1, otherString, otherFPos + 1, otherNumChars);
000720          SELF.StopEdit;
000721
000722          SELF.UpdateRuns(fPos, numChars, otherNumChars);
000723          {$IFC fTrace}EP;{$ENDC}
000724      END;
000725
000726
000727  {$S TK2Start}
000728      PROCEDURE TParagraph.SetTypeStyle(tStyle: TTypeStyle);
000729      BEGIN
000730          {$IFC fTrace}BP(10);{$ENDC}
000731          SetQDTypeStyle(tStyle);
000732          {$IFC fTrace}EP;{$ENDC}
000733      END;
000734
000735
000736  {$S SgTxtHot}
000737      PROCEDURE TParagraph.StyleAt(lp: INTEGER; VAR typeStyle: TTypeStyle);
000738      VAR styleChange:    TStyleChange;
000739          styles:         TArray;
000740          styleIndex:     INTEGER;
000741      BEGIN
000742          {$IFC fTrace}BP(9);{$ENDC}
000743          styles := SELF.typeStyles;
000744          styleIndex := 1;
000745          styles.GetAt(1, @styleChange);
000746          WHILE styleChange.lp <= lp DO
000747              BEGIN
000748              typeStyle := styleChange.newStyle;
000749              styleIndex := styleIndex+1;
000750              styles.GetAt(styleIndex, @styleChange);
000751              END;
000752          {$IFC fTrace}EP;{$ENDC}
000753      END;
000754
000755
000756  {$S SgTxtHot}
000757      PROCEDURE TParagraph.UpdateRuns(atLP: INTEGER; replacedChars: INTEGER; insertedChars: INTEGER);
000758      VAR tStyles:        TArray;
000759          i:              INTEGER;
000760          aChange:        TStyleChange;
000761          tempChange:     TStyleChange;
000762          prevStyle:      TTypeStyle;
000763          lastDeleted:    INTEGER;
```

```
000764      BEGIN
000765          {$IFC fTrace}BP(8);{$ENDC}
000766          tStyles := SELF.typeStyles;
000767          lastDeleted := -1;
000768          i := 1;
000769          WHILE i <= tStyles.size DO
000770              BEGIN
000771              tStyles.GetAt(i, @aChange);
000772              IF atLP <= aChange.lp THEN
000773                  BEGIN
000774                  IF i < tStyles.size THEN
000775                      aChange.lp := aChange.lp - replacedChars;
000776
000777                  {if we deleted some chars, we must delete associated run info}
000778                  IF aChange.lp <= atLP THEN
000779                      BEGIN
000780                      {save type style since we may have deleted only part of this run}
000781                      tempChange := aChange;
000782
000783                      {assume whole run deleted, reinsert later if not}
000784                      tStyles.DelAt(i);
000785                      lastDeleted := i;
000786                      i := i-1;
000787                      END
000788                  ELSE
000789                      BEGIN
000790                      IF i = lastDeleted THEN
000791                          {put back run info for last run deleted if part of it still remains and is not
000792                           the same as the run before the changes}
000793                          IF (aChange.lp <> atLP) AND
000794                              (TFakeTStyle(tempChange.newStyle) <> TFakeTStyle(prevStyle)) THEN
000795                              BEGIN
000796                              tempChange.lp := atLP + insertedChars;
000797                              tStyles.InsAt(i, @tempChange);
000798                              i := i+1;
000799                              END;
000800                      IF i < tStyles.size THEN
000801                          BEGIN
000802                          aChange.lp := aChange.lp + insertedChars;
000803                          tStyles.PutAt(i, @aChange);
000804                          END;
000805                      END;
000806                  END
000807              ELSE
000808                  prevStyle := aChange.newStyle;
000809              i := i+1;
000810              END;
000811          {$IFC fTrace}EP;{$ENDC}
```

```
000812      END;
000813
000814
000815  {$S TK2Start}
000816      FUNCTION TParagraph.Width(i: LONGINT; howMany: INTEGER): INTEGER;
000817      VAR theWidth:   INTEGER;
000818          dumIndex:   INTEGER;
000819      BEGIN
000820          {$IFC fTrace}BP(10);{$ENDC}
000821          dumIndex := 1;
000822          SELF.DrawLine(i-1, i + howMany - 2, FALSE, TRUE, theWidth, dumIndex);
000823          Width := theWidth;
000824          {$IFC fTrace}EP;{$ENDC}
000825      END;
000826
000827
000828  {$S SgTxtIni}
000829  BEGIN
000830      fParaTrace := FALSE;
000831  END;
000832
000833  METHODS OF TEditPara;
000834
000835  {$S SgTxtCld}
000836      FUNCTION TEditPara.CREATE(object: TObject; heap: THeap; initialSize: INTEGER;
000837                                              itsFormat: TParaFormat): TEditPara;
000838      VAR imgList:    TList;
000839      BEGIN
000840          {$IFC fTrace}BP(10);{$ENDC}
000841          IF object = NIL THEN
000842              object := NewDynObject(heap, THISCLASS, initialSize);
000843          SELF := TEditPara(TParagraph.CREATE(object, heap, initialSize, itsFormat.dfltTStyle));
000844          imgList := TList.CREATE(NIL, heap, 0);
000845          WITH SELF DO
000846              BEGIN
000847              bsCount := 0;
000848              nestLevel := 0;
000849              format := itsFormat;
000850              beingFiltered := FALSE;
000851              (*
000852              numImages := 0;
000853              maxImages := 1;
000854              images[1] := NIL;
000855              *)
000856              images := imgList;
000857              END;
000858          itsFormat.ChangeRefCountBy(1);
000859          {$IFC fTrace}EP;{$ENDC}
```

```
000860      END;
000861
000862  {$S SgTxtCld}
000863      PROCEDURE TEditPara.Free;
000864      BEGIN
000865          {$IFC fTrace}BP(10);{$ENDC}
000866          SELF.format.ChangeRefCountBy(-1);
000867          SELF.images.FreeObject; {Free the list, but not its members}
000868          SUPERSELF.Free;
000869          {$IFC fTrace}EP;{$ENDC}
000870      END;
000871
000872
000873  {$S SgTxtCld}
000874  {$IFC fParaTrace}
000875      PROCEDURE TEditPara.Fields(PROCEDURE Field(nameAndType: S255));
000876      VAR str:    STR255;
000877      BEGIN
000878          SUPERSELF.Fields(Field);
000879          Field('bsCount: INTEGER');
000880          Field('nestLevel: INTEGER');
000881          Field('format: TParaFormat');
000882          Field('beingFiltered: BOOLEAN');
000883          (*
000884          Field('maxImages: INTEGER');
000885          Field('numImages: INTEGER');
000886          IntToStr(SELF.numImages, @str);
000887          Field(CONCAT('images: ARRAY[1..', CONCAT(str, '] OF TParaImage')));
000888          *)
000889          Field('images: TList');
000890          Field('');
000891      END;
000892  {$ENDC}
000893
000894  {$S SgTxtHot}
000895      PROCEDURE TEditPara.BeginInsertion(atLP: INTEGER; size:INTEGER);
000896      {Changes the text buffer so that the empty space is located at position atLP;
000897       expands the buffer (if necessary) so that there is at least size empty characters.
000898       (size = 0 means about to backspace; this does nothing if the paragraph is already
000899          setup to backspace at atLP.)
000900      }
000901      BEGIN
000902      {$IFC fTrace} BP(6); {$ENDC}
000903      IF (atLP <> SELF.holeStart) OR (size <> 0) THEN {nothing to do--must be backspacing}
000904          BEGIN
000905          SELF.EditAt(atLP + 1, size);
000906          SELF.bsCount := 0;
000907          END;
```

```
000908      {$IFC fTrace} EP; {$ENDC}
000909      END;
000910
000911
000912  {$S SgTxtCld}
000913      PROCEDURE TEditPara.DelImage(delImage: TParaImage; fFree: BOOLEAN);
000914      BEGIN
000915          {$IFC fTrace}BP(10);{$ENDC}
000916          SELF.images.DelObject(delImage, fFree);
000917          {$IFC fTrace}EP;{$ENDC}
000918      END;
000919
000920
000921  {$S SgTxtCld}
000922      {Selectively delete paraImages from list based on Function Parameter}
000923      PROCEDURE TEditPara.DelImgIF(FUNCTION ShouldDelete(paraImage: TParaImage): BOOLEAN);
000924      VAR s:          TListScanner;
000925          paraImage:  TParaImage;
000926          (*
000927          i:          INTEGER;
000928          numDeleted: INTEGER;
000929          *)
000930      BEGIN
000931          {$IFC fTrace}BP(10);{$ENDC}
000932          s := SELF.images.Scanner;
000933          WHILE s.Scan(paraImage) DO
000934              IF ShouldDelete(paraImage) THEN
000935                  s.Delete(FALSE);
000936          (*
000937          numDeleted := 0;
000938          WITH SELF DO
000939              BEGIN
000940              i := 1;
000941              WHILE i <= numImages DO
000942                  BEGIN
000943                  {$R-} {$H-}
000944                  IF ShouldDelete(images[i]) THEN
000945                      numDeleted := numDeleted+1
000946                  ELSE IF numDeleted > 0 THEN
000947                      images[i-numDeleted] := images[i];
000948                  {$IFC fRngText}{$R+}{$ENDC} {$H+}
000949                  i := i+1;
000950                  END;
000951              IF numDeleted > 0 THEN
000952                  BEGIN
000953                  FOR i := (numImages-numDeleted+1) TO numImages DO
000954                      {$R-}
000955                      images[numImages] := NIL;
```

```
000956                  {$IFC fRngText}{$R+}{$ENDC}
000957              numImages := numImages-numDeleted;
000958              END;
000959          END;
000960      *)
000961      {$IFC fTrace}EP;{$ENDC}
000962  END;
000963
000964
000965  {$S SgTxtCld}
000966  PROCEDURE TEditPara.EachImage(PROCEDURE ImageProc(paraImage: TParaImage));
000967  (*
000968  VAR i:  INTEGER;
000969  *)
000970      PROCEDURE DoIt(object: TObject);
000971      BEGIN
000972          ImageProc(TParaImage(object));
000973      END;
000974  BEGIN
000975      {$IFC fTrace}BP(10);{$ENDC}
000976      SELF.images.Each(DoIt);
000977      (*
000978      FOR i := 1 TO SELF.numImages DO
000979       {$R-}  ImageProc(SELF.images[i]);  {$IFC fRngText}{$R+}{$ENDC}
000980      *)
000981      {$IFC fTrace}EP;{$ENDC}
000982  END;
000983
000984
000985  {$S SgTxtHot}
000986  PROCEDURE TEditPara.EndInsertion;
000987      {After calling this:
000988          holeStart = emptyPos = # chars in paragraph
000989      }
000990  BEGIN
000991      {$IFC fTrace}BP(6);{$ENDC}
000992      SELF.StopEdit;
000993      SELF.bsCount := 0;
000994      {$IFC fTrace}EP;{$ENDC}
000995  END;
000996
000997
000998  {$S SgTxtHot}
000999  FUNCTION TEditPara.GrowSize: INTEGER;
001000  BEGIN
001001      {$IFC fTrace}BP(6);{$ENDC}
001002      GrowSize := 200;
001003      {$IFC fTrace}EP;{$ENDC}
```

```
001004      END;
001005
001006
001007  {$S SgTxtHot}
001008      PROCEDURE TEditPara.InsertOneChar(ch: CHAR; atLP: INTEGER);
001009      BEGIN
001010          {$IFC fTrace}BP(10);{$ENDC}
001011          SELF.BeginInsertion(atLP, 1);      {UNDO}
001012          { now we have SELF.holeStart = atLP }
001013          SELF.PutAt(atLP+1, ch);
001014          SELF.UpdateRuns(atLP, 0, 1);
001015          {$IFC fTrace}EP;{$ENDC}
001016      END;
001017
001018  {$S SgTxtCld}
001019      PROCEDURE TEditPara.InsImage(paraImage: TParaImage);
001020      (*
001021      VAR i:      INTEGER;
001022          found:  BOOLEAN;
001023      *)
001024      BEGIN
001025          {$IFC fTrace}BP(10);{$ENDC}
001026          SELF.images.InsLast(paraImage);
001027          (*
001028          IF SELF.numImages = SELF.maxImages THEN
001029              BEGIN
001030              SELF.ResizeCollection(SELF.size + SELF.holeSize + 4);
001031              SELF.ShiftCollection(0, 4, SELF.size + SELF.holeSize);
001032              WITH SELF DO
001033                  BEGIN
001034                  dynStart := dynStart + 4;
001035                  maxImages := maxImages + 1;
001036                  END;
001037              END;
001038          WITH SELF DO
001039              BEGIN
001040              numImages := numImages + 1;
001041              {$R-}
001042              images[numImages] := paraImage;
001043              {$IFC fRngText}{$R+}{$ENDC}
001044              END;
001045          *)
001046          {$IFC fTrace}EP;{$ENDC}
001047      END;
001048
001049
001050  {$S SgTxtHot}
001051      PROCEDURE TEditPara.SetTypeStyle(tStyle: TTypeStyle);
```

```
001052      BEGIN
001053          {$IFC fTrace}BP(10);{$ENDC}
001054          SELF.format.SetTypeStyle(tStyle);
001055          {$IFC fTrace}EP;{$ENDC}
001056      END;
001057
001058  {$S SgTxtIni}
001059  END;
001060
001061
001062  METHODS OF TLineInfo;
001063
001064  {$S SgTxtWrm}
001065      FUNCTION TLineInfo.CREATE(object: TObject; heap: THeap): TLineInfo;
001066      BEGIN
001067          {$IFC fTrace}BP(6);{$ENDC}
001068          IF object = NIL THEN
001069              object := NewObject(heap, THISCLASS);
001070          SELF := TLineInfo(object);
001071          WITH SELF DO
001072              BEGIN
001073              valid := FALSE;
001074              startLP := 0;
001075              lastDrawnLP := 0;
001076              endLP := 0;
001077              lineLRect := zeroLRect;
001078              lineAscent := 0;
001079              END;
001080          {$IFC fTrace}EP;{$ENDC}
001081      END;
001082
001083  {$S SgTxtCld}
001084  {$IFC fParaTrace}
001085      PROCEDURE TLineInfo.Fields(PROCEDURE Field(nameAndType: S255));
001086      BEGIN
001087          SUPERSELF.Fields(Field);
001088          Field('valid: BOOLEAN');
001089          Field('startLP: INTEGER');
001090          Field('lastDrawnLP: INTEGER');
001091          Field('endLP: INTEGER');
001092          Field('lineLRect: LRect');
001093          Field('lineAscent: INTEGER');
001094          Field('');
001095      END;
001096  {$ENDC}
001097
001098  {$S SgTxtHot}
001099      FUNCTION TLineInfo.LeftCoord(proposedLeftPixel: LONGINT): LONGINT;
```

```
001100      BEGIN
001101          {$IFC fTrace}BP(9);{$ENDC}
001102          {Default is to not change the parameter; TLineInfo subclassers may choose to do otherwise}
001103          LeftCoord := proposedLeftPixel;
001104          {$IFC fTrace}EP;{$ENDC}
001105      END;
001106
001107
001108  {$S SgTxtHot}
001109      FUNCTION TLineInfo.RightCoord(proposedRightPixel: LONGINT): LONGINT;
001110      BEGIN
001111          {$IFC fTrace}BP(9);{$ENDC}
001112          {Default is to not change the parameter; TLineInfo subclassers may choose to do otherwise}
001113          RightCoord := proposedRightPixel;
001114          {$IFC fTrace}EP;{$ENDC}
001115      END;
001116
001117
001118  {$S SgTxtIni}
001119  END;
001120
001121
001122  METHODS OF TParaImage;
001123
001124  {$S SgTxtWrm}
001125      FUNCTION TParaImage.CREATE(object: TObject; heap: THeap; itsView: TView; itsParagraph: TEditPara;
001126                                 itsLRect: LRect; lineTop: LONGINT; lineLeft: LONGINT): TParaImage;
001127
001128      VAR aLineList:  TList;
001129          lineInfo:   TLineInfo;
001130      BEGIN
001131          {$IFC fTrace}BP(10);{$ENDC}
001132          IF object = NIL THEN
001133              object := NewObject(heap, THISCLASS);
001134          SELF := TParaImage(TImage.CREATE(object, heap, itsLRect, itsView));
001135          SELF.paragraph := itsParagraph;
001136          SELF.extentLRect := itsLRect;
001137          aLineList := TList.CREATE(NIL, heap, 0);
001138          lineInfo := SELF.DfltLineInfo(lineTop, lineLeft);
001139          aLineList.InsLast(lineInfo);
001140          WITH SELF DO
001141              BEGIN
001142              height := lineInfo.lineLRect.bottom - lineInfo.lineLRect.top;
001143              lineList := aLineList;
001144              tickcount := 0;
001145              changed := TRUE;
001146              startLP := 0;
001147              endLP := 0;
```

```
001148              textImage := NIL;
001149              wasOffset := FALSE;
001150              END;
001151
001152          {$IFC fTrace}EP;{$ENDC}
001153      END;
001154
001155
001156  {$S SgTxtWrm}
001157      PROCEDURE TParaImage.Free;
001158      BEGIN
001159          {$IFC fTrace}BP(10);{$ENDC}
001160          Free(SELF.lineList);
001161          (* Since caller of this may be scanning the paragraph's image list we can't delete it from the
001162             list here lest we screw up the caller's scanner.  So the caller will have to to do this}
001163          SELF.paragraph.DelImage(SELF, FALSE);
001164          *)
001165          SUPERSELF.Free;
001166          {$IFC fTrace}EP;{$ENDC}
001167      END;
001168
001169  {$S SgTxtCld}
001170  {$IFC fParaTrace}
001171      PROCEDURE TParaImage.Fields(PROCEDURE Field(nameAndType: S255));
001172      BEGIN
001173          SUPERSELF.Fields(Field);
001174          Field('paragraph: TEditPara');
001175          Field('height: INTEGER');
001176          Field('lineList: TList');
001177          Field('changed: BOOLEAN');
001178          Field('tickCount: INTEGER');
001179          Field('startLP: INTEGER');
001180          Field('endLP: INTEGER');
001181          Field('textImage: TTextImage');
001182          Field('wasOffset: BOOLEAN');
001183          Field('');
001184      END;
001185  {$ENDC}
001186
001187  {$S SgTxtHot}
001188      PROCEDURE TParaImage.AdjustLineLPs(atLP, deltaLP: INTEGER);
001189          {positive deltaLP implies character(s) were inserted, negative deltaLP implies they were deleted}
001190          PROCEDURE AdjustLP(obj: TObject);
001191          BEGIN
001192              WITH TLineInfo(obj) DO
001193                  BEGIN
001194                  {$H-}
001195                  IF startLP > atLP THEN
```

```
001196                      startLP := Max(atLP, startLP + deltaLP);
001197                  IF lastDrawnLP > atLP THEN
001198                      lastDrawnLP := Max(atLP, lastDrawnLP + deltaLP);
001199                  IF endLP > atLP THEN
001200                      endLP := Max(atLP, endLP + deltaLP);
001201                  {$H+}
001202                  END;
001203          END;
001204      BEGIN
001205          {$IFC fTrace}BP(10);{$ENDC}
001206          SELF.lineList.Each(AdjustLP);
001207          WITH SELF DO
001208              BEGIN
001209              {$H-}
001210              IF startLP > atLP THEN
001211                  startLP := Max(atLP, startLP + deltaLP);
001212              IF endLP >= atLP THEN
001213                  endLP := Max(atLP, endLP + deltaLP);
001214              {$H+}
001215              END;
001216          {$IFC fTrace}EP;{$ENDC}
001217      END;
001218
001219
001220  {$S SgTxtHot}
001221      PROCEDURE TParaImage.ComputeLineInfo(curLine: TLineInfo; maxLineLen: INTEGER;
001222                                                  VAR nextLP: INTEGER; VAR lRectNeeded: LRect);
001223      BEGIN
001224          {$IFC fTrace}BP(10);{$ENDC}
001225          {$IFC fTrace}EP;{$ENDC}
001226      END;
001227
001228
001229  {$S SgTxtWrm}
001230      FUNCTION TParaImage.DfltLineInfo(lineTop: LONGINT; lineLeft: LONGINT): TLineInfo;
001231      VAR lineInfo:   TLineInfo;
001232          fInfo:      FontInfo;
001233          i:          INTEGER;
001234          format:     TParaFormat;
001235      BEGIN
001236          {$IFC fTrace}BP(9);{$ENDC}
001237          lineInfo := TLineInfo.CREATE(NIL, SELF.Heap);
001238
001239          format := SELF.GetFormat;
001240          format.SetTypeStyle(format.dfltTStyle);
001241          GetFontInfo(fInfo);
001242
001243          i := SELF.paragraph.size;
```

```
001244          WITH lineInfo, fInfo, format DO
001245              BEGIN
001246              lastDrawnLP := i;
001247              endLP := i;
001248              lineAscent := ascent;
001249              {$H-}
001250              SetLRect(lineLRect, lineLeft, lineTop, lineLeft,
001251                          spaceAbovePara + lineTop + ascent + descent + leading + spaceBelowPara);
001252              OffsetLRect(lineLRect, SELF.extentLRect.left + firstIndent, SELF.extentLRect.top);
001253              {$H+}
001254              END;
001255          DfltLineInfo := lineInfo;
001256          {$IFC fTrace}EP;{$ENDC}
001257      END;
001258
001259
001260  {$S SgTxtHot}
001261      PROCEDURE TParaImage.DrawLine(startLP: INTEGER; fDraw: BOOLEAN; stopWidth, wrapWidth: INTEGER;
001262                                      VAR lineWidth, lastToDraw, endLP: INTEGER);
001263      {Figures out what characters to draw based on a variety of input constraints.
001264              Returns:
001265                  lineWidth:  the width of the line calculated (including trailing spaces) [??]
001266                  lastToDraw: the lp of the last non blank character in the line
001267                  endLP:      the lp of the last character in the line (may be a blank)          }
001268
001269      {NOTE: the wrapWidth parameter may eventually be dropped and instead calculated from SELF.extentLRect
001270              and certain format fields}
001271
001272      LABEL   1;
001273      VAR c:              CHAR;
001274          startPP:        INTEGER;
001275          curIndex:       INTEGER;       { PP of last character looked at }
001276          styleChange:    TStyleChange;
001277          prevChange:     TStyleChange;
001278          styles:         TArray;
001279          styleIndex:     INTEGER;
001280          firstStyleIndex:INTEGER;
001281          cWidth:         INTEGER;
001282          maxPP:          INTEGER;
001283          endPP:          INTEGER;
001284          breakIndex:     INTEGER;
001285          breakCount:     INTEGER;
001286          breakLen:       INTEGER;
001287          paragraph:      TEditPara;
001288          format:         TParaFormat;
001289          drawStart:      INTEGER;
001290          drawCount:      INTEGER;
001291          dummy:          INTEGER;
```

```
001292          maxLP:          INTEGER;
001293      BEGIN
001294          {$IFC fTrace}BP(10);{$ENDC}
001295          {$IFC fParaTrace}
001296          IF fParaTrace THEN
001297              writeln('** DrawLine: startLP=', startLP:1, ', maxLP=',maxLP:1, ', fDraw=', fDraw,
001298                      ', stopWidth=',stopWidth:1, ', wrapWidth=',wrapWidth:1);
001299          {$ENDC}
001300
001301          maxLP := SELF.paragraph.size-1;
001302          IF maxLP < startLP THEN
001303              BEGIN
001304              lineWidth := 0;
001305              lastToDraw := maxLP;
001306              endLP := maxLP;
001307              END
001308          ELSE
001309              BEGIN
001310              paragraph := SELF.paragraph;
001311              format := SELF.GetFormat;
001312
001313              breakIndex := 0;
001314              lastToDraw := 0;
001315              lineWidth := 0;
001316              cWidth := 0;
001317              curIndex := paragraph.FixLP(startLP);
001318              styles := paragraph.typeStyles;
001319              styles.GetAt(1, @styleChange);
001320              styleIndex := 1;
001321              REPEAT
001322                  prevChange := styleChange;
001323                  styleIndex := styleIndex + 1;
001324                  styles.GetAt(styleIndex, @styleChange);
001325              UNTIL curIndex < styleChange.lp;
001326
001327              format.SetTypeStyle(prevChange.newStyle);
001328              firstStyleIndex := styleIndex-1;
001329
001330              startPP := curIndex;
001331              maxPP := MIN(paragraph.size, paragraph.FixLP(maxLP));
001332
001333              {$IFC fParaTrace}
001334              IF fParaTrace THEN
001335                  writeln('** DrawLine: About enter loop, maxPP =',maxPP:1,'  holeStart=',paragraph.holeStart:1,
001336                          '  holeSize=',paragraph.holeSize:1);
001337              {$ENDC}
001338              WHILE curIndex <= maxPP DO
001339                  BEGIN
```

```
001340                  IF curIndex = styleChange.lp THEN
001341                      BEGIN
001342                      format.SetTypeStyle(styleChange.newStyle);
001343                      styleIndex := styleIndex+1;
001344                      styles.GetAt(styleIndex, @styleChange)
001345                      END;
001346
001347                  c := paragraph.At(curIndex+1);
001348
001349                  cWidth := CharWidth(c);
001350                  {$IFC fParaTrace AND FALSE}
001351                  IF fParaTrace THEN
001352                      writeln('curIndex=', curIndex:1, ', char=', c, ', cWidth=',
001353                                  cWidth:1, ', lineWidth=',lineWidth:1);
001354                  {$ENDC}
001355
001356
001357                  {Drop out of loop if lineWidth > stopWidth unless
001358                      we're at end of line and have trailing spaces}
001359                  IF (lineWidth + cWidth > stopWidth) THEN
001360                      IF format.wordWrap AND (stopWidth = wrapWidth) THEN
001361                          IF (c <> ' ') THEN
001362                              GOTO 1
001363                          ELSE
001364                      ELSE
001365                          GOTO 1;
001366
001367                  IF format.wordWrap AND (c = ' ') THEN
001368                      BEGIN
001369                      IF (breakIndex + 1) < curIndex THEN  {so we don't draw trailing blanks}
001370                          lastToDraw := curIndex-1;
001371                      breakIndex := curIndex;
001372                      breakLen := lineWidth;
001373                      END;
001374
001375                  lineWidth := lineWidth + cWidth;
001376                  curIndex := curIndex + 1;
001377                  cWidth := 0;
001378                  END;
001379      1:
001380              curIndex := curIndex - 1;
001381              IF format.wordWrap AND (lineWidth + cWidth > wrapWidth) AND (breakIndex > 0) THEN
001382                  { PRIMITIVE WORD WRAP! }
001383                  BEGIN
001384                  lineWidth := breakLen;
001385                  curIndex := breakIndex;
001386                  END
001387              ELSE
```

```
001388                  lastToDraw := curIndex;
001389
001390              {$IFC fParaTrace}
001391              IF fParaTrace THEN
001392                  writeln('** DrawLine: About to figure endLP, curIndex =',curIndex:1);
001393              {$ENDC}
001394
001395              endLP := curIndex;
001396
001397              {$IFC fParaTrace}
001398              IF fParaTrace THEN
001399                  writeln('** DrawLine: endLP figured =',endLP:1);
001400              {$ENDC}
001401
001402              IF (lastToDraw >= 0) AND fDraw THEN
001403                  SELF.FastDrawLine(paragraph.fixLP(startLP), lastToDraw, TRUE,
001404                                                          FALSE, dummy, firstStyleIndex);
001405              END;
001406
001407          {$IFC fParaTrace}
001408          IF fParaTrace THEN
001409              BEGIN
001410              writeln('** DrawLine done: endLP=',endLP:1,', lineWidth=',lineWidth:1);
001411              writeln('** DrawLine done: final lastToDraw=',lastToDraw:1);
001412              WriteLn;
001413              END;
001414          {$ENDC}
001415          {$IFC fTrace}EP;{$ENDC}
001416      END;
001417
001418
001419  {$S SgTxtHot}
001420      PROCEDURE TParaImage.DrawParaImage(limitLRect: LRect; startLP: INTEGER; drawAction: TDrawAction;
001421                                              invalBits: BOOLEAN; VAR drawnLRect: LRect);
001422      {Note: DrawParaImage now assumes that the paragraph was changed}
001423      LABEL 1;
001424      VAR paragraph:      TEditPara;
001425          fInfo:          FontInfo;
001426          lineInfo:       TLineInfo;
001427          firstLineInfo:  TLineInfo;
001428          lineList:       TList;
001429          lineSpacing:    INTEGER;
001430          curBase:        LONGINT;
001431          leftMargin:     LONGINT;
001432          curLP:          INTEGER;
001433          endLP:          INTEGER;
001434          numChars:       INTEGER;
001435          lineLen:        INTEGER;
```

```
001436          maxLineLen:     INTEGER;
001437          lineIndex:      INTEGER;
001438          pixel:          LONGINT;
001439          lastDrawnLP:    INTEGER;
001440          testLPoint:     LPoint;
001441          format:         TParaFormat;
001442          firstFudge:     INTEGER;
001443          startOfNewPara: BOOLEAN;
001444          anLRect:        LRect;
001445          sLine:          TListScanner;
001446          genRest:        BOOLEAN;
001447          genBefore:      BOOLEAN;
001448          oldEndLP:       INTEGER;
001449          prevLineInfo:   TLineInfo;
001450          prevLen:        INTEGER;
001451          prevPImage:     TParaImage;
001452          prevTxtImage:   TTextImage;
001453          origStart:      INTEGER;
001454          dummy:          INTEGER;
001455          styleIndex:     INTEGER;
001456          r:              LRect;
001457          {$IFC fParaTrace}
001458          str:            STR255;
001459          {$ENDC}
001460      BEGIN
001461          {$IFC fTrace}BP(10);{$ENDC}
001462          {$IFC fParaTrace}
001463          IF fParaTrace THEN
001464              BEGIN
001465              WITH limitLRect DO
001466                  WriteLn('## Entering DrawParaImage: limitLRect=[(',left:1,',',top:1,'),(',
001467                                                                  right:1,',', bottom:1,')]');
001468              LIntToHex(ORD(SELF), @str);
001469              WriteLn(    '       SELF = ', str,' startLP=',startLP:1,'  drawAction=',ORD(drawAction));
001470              END;
001471          {$ENDC}
001472
001473          drawnLRect := limitLRect;
001474          endLP := startLP;
001475          paragraph := SELF.paragraph;
001476          format := SELF.GetFormat;
001477          IF drawAction = actionDraw THEN WITH SELF DO
001478              tickCount := (tickCount+1) MOD MAXINT;
001479
001480          PicGrpBegin;
001481          PenNormal;
001482
001483          genRest := FALSE;
```

```
001484          genBefore := FALSE;
001485          curLP := startLP;
001486          numChars := paragraph.size;
001487          SELF.startLP := curLP;
001488
001489          format.SetTypeStyle(format.dfltTStyle);
001490          GetFontInfo(fInfo);
001491          WITH fInfo DO
001492              lineSpacing := ascent + descent + leading + format.lineSpacing;
001493
001494          lineList := SELF.lineList;
001495          curBase := limitLRect.top;
001496          prevLineInfo := NIL;
001497          IF lineList.Size > 0 THEN
001498              BEGIN
001499              sLine := lineList.Scanner;
001500              {If existing lineInfo's start after startLP then we need to generate preceeding lineInfo's}
001501              IF TLineInfo(lineList.First).startLP > startLP THEN
001502                  BEGIN
001503                  origStart := TLineInfo(lineList.First).startLP;
001504                  genBefore := TRUE;
001505                  END
001506              ELSE WHILE sLine.Scan(lineInfo) DO
001507                  BEGIN
001508                  {delete lineinfo's that start before the startLP parameter}
001509                  IF lineInfo.endLP < startLP THEN
001510                      sLine.Delete(TRUE)
001511                  ELSE
001512                      BEGIN
001513                      IF lineInfo.valid THEN
001514                          BEGIN
001515                          prevLineInfo := lineInfo;
001516                          curBase := lineInfo.lineLRect.bottom;
001517                          END
001518                      ELSE
001519                          GOTO 1;
001520                      END;
001521                  END;
001522              END
001523          ELSE
001524              lineInfo := NIL;
001525
001526          1:
001527          IF NOT genBefore THEN
001528              IF lineInfo = NIL THEN
001529                  BEGIN
001530                  genRest := TRUE;
001531                  curLP := Max(startLP, SELF.endLP);
```

```
001532              END
001533          ELSE
001534              curLP := Max(startLP, lineInfo.startLP);
001535
001536      startOfNewPara := curLP = 0;
001537
001538      curBase := curBase + fInfo.ascent;
001539      IF startOfNewPara THEN
001540          BEGIN
001541          curBase := curBase + format.spaceAbovePara;
001542          leftMargin := limitLRect.left + format.firstIndent;
001543          { The first line maxLineLen might be different (due to firstIndent)}
001544          firstFudge := format.firstIndent - format.leftIndent;
001545          END
001546      ELSE
001547          BEGIN
001548          leftMargin := limitLRect.left + format.leftIndent;
001549          firstFudge := 0;
001550          END;
001551
001552      limitLRect.left := limitLRect.left + format.leftIndent;
001553      limitLRect.right := limitLRect.right - format.rightIndent;
001554
001555      maxLineLen := lengthLRect(limitLRect, h) - firstFudge;  {if firstIndent is to left of
001556                                                               leftIndent, fudge will be negative}
001557
001558      SetLPt(testLPoint, limitLRect.left, curBase + fInfo.descent);
001559
001560      {$IFC fParaTrace}
001561      IF fParaTrace THEN
001562          BEGIN
001563          WriteLn('## DrawParaImage: Entering DrawLine loop -- leftMargin=',leftMargin:1,
001564                          '  curBase=',curBase:1);
001565          END;
001566      {$ENDC}
001567
001568      {don't bother going into the loop if we can't fit the first line or we've run out
001569       of characters already (eg: empty para)}
001570      IF (NOT LPtInLRect(testLPoint, limitLRect)) OR (curLP >= numChars) THEN
001571          BEGIN
001572          SELF.extentLRect := drawnLRect;
001573          IF NOT genRest THEN
001574              BEGIN
001575              r := lineInfo.lineLRect;
001576              r.left := lineInfo.LeftCoord(SELF.textImage.extentLRect.left-1);
001577              r.right := lineInfo.RightCoord(SELF.textImage.extentLRect.right+1);
001578              lineInfo := SELF.DfltLineInfo(0,0);
001579              r.bottom := lineInfo.lineLRect.bottom;
```

```
001580              sLine.Replace(lineInfo, TRUE);
001581              genRest := NOT sLine.Scan(lineInfo);
001582              END;
001583          {These two assignments distinguish (for the calling routine) whether an empty
001584           paragraph did or did not fit in the limitLRect.  Assuming curLP = 0, SELF.endLP
001585           will be 0 for a paragraph that did fit and -1 for one that did not fit.  The calling
001586           routine checks this value against paragraph.size to see if the paragraph fit}
001587          IF LPtInLRect(testLPoint, limitLRect) THEN
001588              BEGIN
001589              SELF.endLP := curLP;
001590              {Erase the old line}
001591              IF drawAction = actionDraw THEN
001592                  FillLRect(r, lPatWhite)
001593              ELSE IF drawAction = actionInval THEN
001594                  thePad.InvalLRect(r);
001595              END
001596          ELSE
001597              SELF.endLP := curLP-1;
001598          {$IFC fParaTrace}
001599          IF fParaTrace THEN
001600              BEGIN
001601              WriteLn('## DrawParaImage: Empty para or cannot fit; endLP set to ',SELF.endLP:1);
001602              END;
001603          {$ENDC}
001604          END
001605      {Otherwise, set up lineLRect and call DrawLine while there are still characters to display
001606       and we still fit in limitLRect}
001607      ELSE
001608          BEGIN
001609          {Layout line previous to first invalid line to see if characters from the
001610           invalid line can wrap back.  First, however, we must check for special case
001611           of the previous line being in another textImage.}
001612          IF NOT startOfNewPara AND (prevLineInfo = NIL) THEN
001613              BEGIN
001614              prevTxtImage := SELF.textImage.prevTxtImg;
001615              IF prevTxtImage <> NIL THEN
001616                  BEGIN
001617                  prevPImage := TParaImage(prevTxtImage.imageList.Last);
001618                  prevLineInfo := TLineInfo(prevPImage.lineList.Last);
001619                  prevLen := LengthLRect(prevPImage.extentLRect, h) - format.leftIndent
001620                                                                    - format.rightIndent;
001621                  END;
001622              END
001623          ELSE
001624              BEGIN
001625              prevLen := maxLineLen;
001626              prevPImage := SELF;
001627              END;
```

```
001628
001629              IF prevLineInfo <> NIL THEN
001630                  BEGIN
001631                  oldEndLP := prevLineInfo.endLP;
001632                  IF prevLineInfo.startLP = 0 THEN
001633                      prevLen := prevLen - (format.firstIndent - format.leftIndent);
001634
001635                  prevPImage.DrawLine(prevLineInfo.startLP, FALSE, prevLen, prevLen,
001636                                                  lineLen, lastDrawnLP, endLP);
001637
001638                  IF endLP <> oldEndLP THEN
001639                      BEGIN
001640                      SELF.textImage.useFirstPixel := FALSE;
001641                      r := prevLineInfo.lineLRect;
001642                      r.left := prevLineInfo.LeftCoord(prevPImage.textImage.extentLRect.left-1);
001643                      r.right := prevLineInfo.RightCoord(prevPImage.textImage.extentLRect.right+1);
001644                      IF drawAction = actionDraw THEN
001645                          BEGIN
001646                          FillLRect(r, lPatWhite);
001647                          styleIndex := 1;
001648                          MoveToL(prevLineInfo.lineLRect.left,
001649                                  prevLineInfo.lineLRect.top+prevLineInfo.lineAscent);
001650                          SELF.FastDrawLine(prevLineInfo.startLP, lastDrawnLP, TRUE,
001651                                                      FALSE, dummy, styleIndex);
001652                          END
001653                      ELSE IF drawAction = actionInval THEN
001654                          thePad.InvalLRect(r);
001655
001656                      WITH prevLineInfo.lineLRect DO
001657                          BEGIN
001658                          prevLineInfo.valid := invalBits;
001659                          right := left + lineLen;
001660                          prevLineInfo.lastDrawnLP := lastDrawnLP;
001661                          prevLineInfo.endLP := endLP;
001662                          curLP := endLP + 1;
001663                          prevPImage.endLP := curLP;
001664                          IF curLP >= numChars THEN
001665                              IF (bottom + format.spaceBelowPara) <= limitLRect.bottom THEN
001666                                  BEGIN
001667                                  r.top := bottom;
001668                                  bottom := bottom + format.spaceBelowPara;
001669                                  r.bottom := bottom;
001670                                  {$H-}
001671                                  IF drawAction = actionDraw THEN
001672                                      FillLRect(r, lPatWhite)
001673                                  ELSE IF drawAction = actionInval THEN
001674                                      thePad.InvalLRect(r);
001675                                  {$H+}
```

```
001676                          END;
001677                      END;
001678                  END;
001679              END;
001680          WITH fInfo DO
001681              SetLRect(anLRect, leftMargin, curBase - ascent,
001682                       leftMargin + maxlineLen, curBase + descent + leading);
001683          leftMargin := limitLRect.left;
001684          {Setup GrafPort for first line (after prev line)}
001685          IF drawAction = actionDraw THEN
001686              MoveToL(leftMargin + firstFudge, curBase);
001687
001688          WHILE (curLP < numChars) AND (LPtInLRect(testLPoint, limitLRect)) DO
001689              BEGIN
001690              IF genRest OR genBefore THEN
001691                  lineInfo := TLineInfo.CREATE(NIL, paragraph.heap);
001692
001693              IF NOT lineInfo.valid THEN
001694                  BEGIN
001695                  WITH fInfo, lineInfo DO
001696                      BEGIN
001697                      startLP := curLP;
001698                      lineAscent := ascent;
001699                      lineLRect := anLRect;
001700                      END;
001701
001702                  r := anLRect;
001703                  r.left := lineInfo.LeftCoord(SELF.textImage.extentLRect.left-1);
001704                  IF SELF.textImage.useFirstPixel THEN
001705                      BEGIN
001706                      r.left := Max(r.left, SELF.textImage.firstLinePixel);
001707                      SELF.textImage.useFirstPixel := FALSE;
001708                      END;
001709                  r.right := lineInfo.RightCoord(SELF.textImage.extentLRect.right+1);
001710
001711                  IF drawAction = actionDraw THEN
001712                      FillLRect(r, lPatWhite)
001713                  ELSE IF drawAction = actionInval THEN
001714                      thePad.InvalLRect(r);
001715
001716                  oldEndLP := lineInfo.endLP;
001717
001718                  SELF.DrawLine(curLP, drawAction = actionDraw, maxLineLen, maxLineLen,
001719                                       lineLen, lastDrawnLP, endLP);
001720
001721                  {$IFC fParaTrace}
001722                  IF (curLP > endLP) AND (curLP < numChars)THEN
001723                      BEGIN
```

```
001724                      ABCbreak('loop in DrawParaImage; curLP=',curLP);
001725                      endLP := curLP + 1;
001726                      END;
001727                  {$ENDC}
001728
001729                  {finish setting up new lineInfo}
001730                  WITH fInfo, lineInfo.lineLRect DO
001731                      BEGIN
001732                      lineInfo.lastDrawnLP := lastDrawnLP;
001733                      lineInfo.endLP := endLP;
001734                      right := left + lineLen;
001735
001736                      {if this is last line in paragraph, add spaceBelowPara, unless that extra amount
001737                       would put it ouside of the limitLRect}
001738                      IF (endLP+1) >= numChars THEN
001739                          IF (bottom + format.spaceBelowPara) <= limitLRect.bottom THEN
001740                              BEGIN
001741                              r.top := bottom;
001742                              bottom := bottom + format.spaceBelowPara;
001743                              r.bottom := bottom;
001744                              {$H-}
001745                              IF drawAction = actionDraw THEN
001746                                  FillLRect(r, lPatWhite)
001747                              ELSE IF drawAction = actionInval THEN
001748                                  thePad.InvalLRect(r);
001749                              {$H+}
001750                              END;
001751                      END;
001752
001753                      {If the word being typed wrapped down to the next line, we need to erase
001754                       the piece of the word that was on this line.}
001755                      IF r.left > lineInfo.lineLRect.right THEN
001756                          BEGIN
001757                          r.left := lineInfo.lineLRect.right;
001758                          IF drawAction = actionDraw THEN
001759                              FillLRect(r, lPatWhite)
001760                          ELSE IF drawAction = actionInval THEN
001761                              thePad.InvalLRect(r);
001762                          END;
001763
001764                      IF genRest THEN
001765                          lineList.InsLast(lineInfo)
001766                      ELSE IF genBefore THEN
001767                          sLine.Append(lineInfo); {leaves scanner poised before the original first lineInfo}
001768
001769                      lineInfo.valid := invalBits;
001770                      END
001771                  ELSE {lineInfo is valid}
```

```
001772                  BEGIN
001773                  endLP := lineInfo.endLP;
001774                  oldEndLP := endLP;
001775                  END;
001776
001777              {This field is used by caller in case the entire paragraph didn't fit, so that the
001778               caller knows where to start subsequent display}
001779              SELF.endLP := endLP + 1;
001780              {Setup for next line}
001781              curLP := endLP+1;
001782              curBase := curBase + lineSpacing;
001783              maxLineLen := maxLineLen + firstFudge;
001784              firstFudge := 0;
001785              IF genBefore THEN
001786                  genBefore :=  origStart > curLP;
001787
001788              IF NOT (genRest OR genBefore) THEN
001789                  BEGIN
001790                  IF sLine.Scan(lineInfo) THEN
001791                      IF lineInfo.startLP <> curLP THEN
001792                          lineInfo.valid := FALSE
001793                      ELSE
001794                  ELSE
001795                      genRest := TRUE;
001796                  END;
001797
001798              { setup GrafPort and lineRect for next line}
001799              IF drawAction = actionDraw THEN
001800                  MoveToL(leftMargin, curBase);
001801              WITH fInfo DO
001802                  SetLRect(anLRect,
001803                          leftMargin, curBase - ascent,
001804                          leftMargin + maxLineLen, curBase + descent + leading);
001805              SetLPt(testLPoint, leftMargin, testLPoint.v + lineSpacing);
001806              END; {WHILE}
001807          END; {IF}
001808      {We don't want to delete the lineInfo we just sLine.appended so advance scanner}
001809      IF genBefore THEN
001810          sLine.Skip(1);
001811      IF NOT genRest THEN
001812          REPEAT
001813              sLine.Delete(TRUE);
001814          UNTIL NOT sLine.Scan(lineInfo);
001815
001816      IF SELF.changed THEN
001817          SELF.changed := NOT invalBits;
001818
001819      PicGrpEnd;
```

```
001820
001821          lineInfo := TLineInfo(SELF.lineList.Last);
001822          firstLineInfo := TLineInfo(SELF.lineList.First);
001823          SELF.height := lineInfo.lineLRect.bottom - firstLineInfo.lineLRect.top;
001824          drawnLRect.bottom := lineInfo.lineLRect.bottom;
001825
001826          SELF.extentLRect := drawnLRect;
001827
001828          {$IFC fParaTrace}
001829          IF fParaTrace THEN
001830              BEGIN
001831              WITH drawnLRect DO
001832                  WriteLn('## Exiting DrawParaImage: drawnLRect=[(',left:1,',',top:1,'),(',
001833                                                                  right:1,',', bottom:1,')]',
001834                                                                  '; height = ',SELF.height:1);
001835              END;
001836          {$ENDC}
001837
001838          {$IFC fTrace}EP;{$ENDC}
001839      END;
001840
001841
001842  {$S SgTxtCld}
001843      PROCEDURE TParaImage.Draw;
001844      BEGIN
001845          {$IFC fTrace}BP(10);{$ENDC}
001846          SELF.RedrawLines(0, MAXINT);
001847          {$IFC fTrace}EP;{$ENDC}
001848      END;
001849
001850
001851  {$S SgTxtHot}
001852      PROCEDURE TParaImage.FastDrawLine(startLP, endLP: INTEGER; fDraw: BOOLEAN; fWidth: BOOLEAN;
001853                                        VAR width: INTEGER; VAR styleIndex: INTEGER);
001854      {If fDraw = TRUE, draws a line of characters from startLP to endLP; does not worry about word wrap.
001855       If fWidth = TRUE, returns width of characters.  Also accepts an initial styleIndex (index into
001856       run array) to make typestyle scanning faster. Returns styleIndex of run of last character drawn.}
001857
001858      VAR saveFormat: TParaFormat;
001859          paragraph:  TEditPara;
001860          format:     TParaformat;
001861      BEGIN
001862          {$IFC fTrace}BP(10);{$ENDC}
001863          paragraph := SELF.paragraph;
001864          saveFormat := paragraph.format;
001865          format := SELF.GetFormat;
001866          paragraph.format := format;
001867          paragraph.DrawLine(startLP, endLP, fDraw, fWidth, width, styleIndex);
```

```
001868          paragraph.format := saveFormat;
001869          {$IFC fTrace}EP;{$ENDC}
001870      END;
001871
001872
001873  {$S SgTxtHot}
001874      {Returns paragraph's paraFormat; can be overriden by application}
001875      FUNCTION TParaImage.GetFormat: TParaFormat;
001876      VAR styleIndex:     INTEGER;
001877      BEGIN
001878          {$IFC fTrace}BP(7);{$ENDC}
001879          GetFormat := SELF.paragraph.format;
001880          {$IFC fTrace}EP;{$ENDC}
001881      END;
001882
001883
001884  {$S SgTxtHot}
001885      PROCEDURE TParaImage.InvalLinesWith(startLP, endLP: INTEGER);
001886      VAR s:              TListScanner;
001887          lineInfo:       TLineInfo;
001888          prevLineInfo:   TLineInfo;
001889          numChars:       INTEGER;
001890      BEGIN
001891          {$IFC fTrace}BP(9);{$ENDC}
001892          SELF.changed := TRUE;
001893          s := SELF.lineList.Scanner;
001894          IF s.Scan(prevLineInfo) THEN
001895              BEGIN
001896              WHILE s.Scan(lineInfo) DO
001897                  BEGIN
001898                  {If its already invalid, don't muck with it}
001899                  IF prevLineInfo.valid THEN
001900                      prevLineInfo.valid := (lineInfo.startLP <= startLP) OR (prevLineInfo.startLP > endLP);
001901                  prevLineInfo := lineInfo;
001902                  END;
001903
001904              {last line}
001905              IF prevLineInfo.valid THEN
001906                  BEGIN
001907                  numChars := SELF.paragraph.size;
001908                  prevLineInfo.valid := (numChars < startLP) OR (prevLineInfo.startLP > endLP);
001909                  END;
001910              END;
001911          {$IFC fTrace}EP;{$ENDC}
001912      END;
001913
001914
001915  {$S SgTxtWrm}
```

```
001916      PROCEDURE TParaImage.LineWithLPt(pt: LPoint; VAR lineIndex: INTEGER; VAR lineInfo: TLineInfo);
001917      VAR s:          TListScanner;
001918          nxtLnInfo:  TLineInfo;
001919      BEGIN
001920          {$IFC fTrace}BP(8);{$ENDC}
001921          {$IFC fParaTrace}
001922          IF fParaTrace THEN
001923              writeln('in LineWithLPt, point=(',pt.h:1,', ',pt.v:1,')');
001924          {$ENDC}
001925
001926          s := SELF.lineList.scanner; {&&& maybe could use TList.scannerFrom(index)}
001927          lineIndex := 1;
001928
001929          {This has been modified to allow for the possibility of multiple lineRects at the same
001930           vertical coordinate }
001931          IF s.Scan(lineInfo) THEN
001932              WHILE s.scan(nxtLnInfo) DO
001933                  BEGIN
001934                  IF (pt.v < lineInfo.lineLRect.bottom) AND
001935                      ((pt.h < lineInfo.lineLRect.right) OR (pt.v < nxtLnInfo.lineLRect.top)) THEN
001936                          s.Done
001937                  ELSE
001938                      BEGIN
001939                      lineIndex := lineIndex + 1;
001940                      lineInfo := nxtLnInfo;
001941                      END;
001942                  END
001943          ELSE
001944              BEGIN
001945              {$IFC fParaTrace}
001946              IF fParaTrace THEN
001947                  writeln(chr(7), 'LineWithLPt: no TLineInfo in TParaImage, lineIndex=0');
001948              {$ENDC}
001949              lineIndex := 1;
001950              lineInfo := SELF.DfltLineInfo(0, 0);
001951              SELF.lineList.InsLast(lineInfo);
001952              END;
001953          {$IFC fTrace}EP;{$ENDC}
001954      END;
001955
001956
001957  {$S SgTxtHot}
001958      PROCEDURE TParaImage.LocateLP(LP: INTEGER; VAR lineIndex: INTEGER; VAR pixel: LONGINT);
001959      VAR s:          TListScanner;
001960          lstLnInfo:  TLineInfo;
001961          lineInfo:   TLineInfo;
001962      BEGIN
001963          {$IFC fTrace}BP(8);{$ENDC}
```

```
001964          IF LP < 0 THEN
001965              LP := 0;
001966          s := SELF.lineList.Scanner;
001967          lineIndex := 0;
001968
001969          WHILE s.Scan(lineInfo) DO
001970              BEGIN
001971              IF LP < lineInfo.startLP THEN
001972                  s.Done
001973              ELSE
001974                  BEGIN
001975                  lineIndex := lineIndex + 1;
001976                  lstLnInfo := lineInfo;
001977                  END;
001978              END;
001979
001980          IF lineIndex=0 THEN
001981              BEGIN
001982              {$IFC fParaTrace}
001983              IF fParaTrace THEN
001984                  writeln(chr(7), 'LocateLP: no TLineInfo in TParaImage, lineIndex=0');
001985              {$ENDC}
001986              lineIndex := 1;
001987              lineInfo := SELF.DfltLineInfo(0, 0);
001988              pixel := lineInfo.lineLRect.left - 1;    { leave 1 pixel space before character }
001989              SELF.lineList.InsLast(lineInfo);
001990              END
001991          ELSE
001992              BEGIN
001993              pixel := lstLnInfo.lineLRect.left + SELF.ParaTextWidth(lstLnInfo.startLP, LP-1) - 1;
001994                                                                  { leave 1 pixel space before character }
001995              END;
001996          {$IFC fTrace}EP;{$ENDC}
001997      END;
001998
001999
002000  {$S SgTxtHot}
002001      FUNCTION TParaImage.LpWithLPt(pt: LPoint): INTEGER;
002002      VAR lineIndex:  INTEGER;
002003          lineInfo:   TLineInfo;
002004          endLP:      INTEGER;
002005          lineLen:    INTEGER;
002006          charWid:    INTEGER;
002007          paragraph:  TEditPara;
002008          wrapMargin: INTEGER;
002009          lastLP:     INTEGER;
002010          PROCEDURE DrawLine(obj: TObject);  {This routine gets filtered after a type style change}
002011          BEGIN
```

```
002012              SELF.DrawLine(lineInfo.startLP, FALSE, pt.h-lineInfo.lineLRect.left, wrapMargin,
002013                                  lineLen, lastLP, endLP);
002014          END;
002015      BEGIN
002016          {$IFC fTrace}BP(8);{$ENDC}
002017          SELF.LineWithLPt(pt, lineIndex, lineInfo);
002018          IF pt.v < lineInfo.lineLRect.top THEN
002019              pt := lineInfo.lineLRect.topLeft
002020          ELSE IF pt.v > lineInfo.lineLRect.bottom THEN
002021              pt := lineInfo.lineLRect.botRight
002022          ELSE
002023              LRectHaveLPt(lineInfo.lineLRect, pt);
002024
002025          paragraph := SELF.paragraph;
002026          wrapMargin := lengthLRect(SELF.extentLRect, h);
002027
002028          SELF.FilterAndDo(SELF, DrawLine);
002029
002030          { endLP is now the LP of the character before the character the cursor was over}
002031          lineLen := lineLen + lineInfo.lineLRect.left;
002032          { lineLen is now the x-coord of screen position of endLP (right pixel) }
002033
002034          endLP := MIN(endLP+1, paragraph.size);
002035          charWid := SELF.paraTextWidth(endLP, endLP);   { find width of the char under cursor }
002036          {$IFC fParaTrace}
002037          IF fParaTrace THEN
002038              writeln('LpWithLPt: endLP=', endLP:1, '  pt.h=', pt.h:1, '  lineLen=', lineLen:1,
002039                              '  charWid=', charWid:1);
002040          {$ENDC}
002041          IF 2*(pt.h-lineLen) >= charWid THEN  { pt is right of center of char }
002042              LpWithLPt := paragraph.fixLP(endLP+1)
002043          ELSE                                 { pt is left of center of char }
002044              LpWithLPt := paragraph.fixLP(endLP);
002045          {$IFC fTrace}EP;{$ENDC}
002046      END;
002047
002048
002049  {$S SgTxtCld}
002050      PROCEDURE TParaImage.OffsetBy(deltaLPt: LPoint);
002051      VAR s:              TListScanner;
002052          lineInfo:       TLineInfo;
002053      BEGIN
002054          {$IFC fTrace}BP(10);{$ENDC}
002055          {&&& should make sure the results falls within view}
002056          WITH deltaLPt DO
002057              BEGIN
002058              {$H-}   OffsetLRect(SELF.extentLRect, h, v);    {$H+}
002059              s := SELF.lineList.Scanner;
```

```
002060              WHILE s.Scan(lineInfo) DO
002061                  {$H-}   OffsetLRect(lineInfo.lineLRect, h, v);    {$H+}
002062              END;
002063          {$IFC fTrace}EP;{$ENDC}
002064      END;
002065
002066
002067  {$S SgTxtHot}
002068      {Returns width of characters in range startLP, endLP. (NOTE: startLP=endLP for width of one char)}
002069      FUNCTION TParaImage.ParaTextWidth(startLP, endLP: INTEGER): INTEGER;
002070      VAR styleIndex:     INTEGER;
002071          width:          INTEGER;
002072          PROCEDURE FastDraw(obj: TObject);  {This routine gets filtered after a type style change}
002073          BEGIN
002074              SELF.FastDrawLine(startLP, endLP, FALSE, TRUE, width, styleIndex);
002075          END;
002076      BEGIN
002077          {$IFC fTrace}BP(8);{$ENDC}
002078          styleIndex := 1;
002079          IF endLP < startLP THEN
002080              width := 0
002081          ELSE
002082              SELF.FilterAndDo(SELF, FastDraw);
002083          ParaTextWidth := width;
002084          {$IFC fTrace}EP;{$ENDC}
002085      END;
002086
002087
002088  {$S SgTxtHot}
002089      PROCEDURE TParaImage.RedrawLines(startLine: INTEGER; endLine: INTEGER);
002090      VAR s:              TListScanner;
002091          i:              INTEGER;
002092          lineInfo:       TLineInfo;
002093          prevLineInfo:   TLineInfo;
002094          styleIndex:     INTEGER;
002095          dummy:          INTEGER;
002096          PROCEDURE FastDraw(obj: TObject);  {This routine gets filtered after a type style change}
002097          BEGIN
002098              SELF.FastDrawLine(lineInfo.startLP, lineInfo.lastDrawnLP, TRUE,
002099                                                                  FALSE, dummy, styleIndex);
002100          END;
002101      BEGIN
002102          {$IFC fTrace}BP(10);{$ENDC}
002103          PicGrpBegin;
002104          s := SELF.lineList.Scanner;
002105          i := 0;
002106          endLine := Min(endLine, SELF.lineList.Size);
002107          startLine := Max(startLine, 1);
```

```
002108          styleIndex := 1;
002109          WHILE s.scan(lineInfo) DO
002110              BEGIN
002111              i := i+1;
002112
002113              IF i < startLine THEN {nothing}
002114              ELSE
002115                  BEGIN
002116                  IF LRectIsVisible(lineInfo.lineLRect) THEN
002117                      BEGIN
002118                      {$IFC fParaTrace}
002119                      IF fParaTrace THEN
002120                          writeln('## ReDrawLines: About to call FastDraw; i=', i:1);
002121                      {$ENDC}
002122                      MoveToL(lineInfo.lineLRect.left, lineInfo.lineLRect.top+lineInfo.lineAscent);
002123                      SELF.FilterAndDo(SELF, FastDraw);
002124                      END;
002125                  IF i = endLine THEN
002126                      s.Done;
002127                  END;
002128              END;
002129          PicGrpEnd;
002130          {$IFC fTrace}EP;{$ENDC}
002131      END;
002132
002133
002134  {$S SgTxtCld}
002135      FUNCTION TParaImage.SeesSameAs(image: TImage): BOOLEAN;
002136      BEGIN
002137          {$IFC fTrace}BP(6);{$ENDC}
002138          IF SELF = image THEN
002139              SeesSameAs := TRUE
002140          ELSE IF NOT InClass(image, TParaImage) THEN
002141              SeesSameAs := FALSE
002142          ELSE
002143              SeesSameAs := SELF.paragraph = TParaImage(image).paragraph;
002144          {$IFC fTrace}EP;{$ENDC}
002145      END;
002146
002147
002148  {$S SgTxtIni}
002149  END;
002150  {$S SgTxtIni}
002151

End of File -- Lines: 2151 Characters: 71585
```

```
================================================================================================
FILE: "LIBTK/UTEXT3.TEXT"
================================================================================================

000001  {UText3}
000002  {TStyleSheet, TText, TTextImage, TTextView, TTextWriteUnivText}
000003
000004  {changed 05/11/84 1135  Changed TTextImage.InvalAll to call panel.InvalLRect}
000005  {changed 04/26/84 1308  Changed TTextWriteUnivText.FillRun to TTextWriteUnivText.FillParagraph}
000006  {changed 04/25/84 1250  Changed FilterAndDo calls back to filtering TParaImage for Compugraphic}
000007  {changed 04/25/84 1135  Do same for TText.MarkChanged as for HiliteParagraphs below}
000008  {changed 04/24/84 1637  Make TText.HiliteParagraphs use selection's textImage to call HiliteText}
000009  {changed 04/20/84 1102  Modified TTextImage.ImageWith so that if it finds an image whose endLP equals
000010                              the passed lp then it may continue the scan, favoring the paraImage whose
000011                              textImage equals SELF}
000012  {changed 04/17/84 1349  Erase/Invalidate bottom of textImage BEFORE calling nextTxtImg.RecomputeImages;
000013                          Fix boundary condition bug in TTextImage.RecomputeImages}
000014  {changed 04/17/84 1110  Numerous additional explicit deletions of paraImages from paragraph.images}
000015  {changed 04/16/84 1539  Set last parameter in NewView call in TTextView.CREATE to FALSE}
000016  {changed 04/16/84 1446  Put picture comments in TTextImage.Draw}
000017  {changed 04/16/84 1015  Explicitly delete paraImages from paragraph.images in TTextImage.RecomputeImages}
000018  {changed 04/13/84 1818  Removed test to see if any paraImages in first textImage in TText.HiliteText and
000019                              TText.MarkChanged}
000020  {changed 04/13/84 1537  Changed calls to FilterAndDo to pass TEditPara rather than TParaImage}
000021  {changed 04/13/84 0209  Added TTextImage.NewEditPara}
000022  {changed 04/10/84 1400  Changed references to TEditPara.images in DelPara, DelImagesWith, and ImageWith}
000023  {changed 04/09/84 1337  Use deferUpdate in DrawOrInval and Recompute to decide if we should draw now}
000024
000025  {$S SgTxtHot}
000026
000027
000028  METHODS OF TStyleSheet;
000029
000030  {$S SgTxtIni}
000031      FUNCTION TStyleSheet.CREATE(object: TObject; heap: THeap): TStyleSheet;
000032      VAR aList:  TList;
000033      BEGIN
000034          {$IFC fTrace}BP(6);{$ENDC}
000035          IF object = NIL THEN
000036              object := NewObject(heap, THISCLASS);
000037          SELF := TStyleSheet(object);
000038          aList := TList.CREATE(NIL, heap, 0);
000039          SELF.formats := aList;
000040          {$IFC fTrace}EP;{$ENDC}
000041      END;
000042
000043  {$S SgTxtCld}
```

```
000044      PROCEDURE TStyleSheet.Free;
000045      BEGIN
000046          {$IFC fTrace}BP(10);{$ENDC}
000047          Free(SELF.formats);
000048          SUPERSELF.Free;
000049          {$IFC fTrace}EP;{$ENDC}
000050      END;
000051
000052
000053  {$S SgTxtIni}
000054      PROCEDURE TStyleSheet.InitDefault;
000055      VAR paraFormat: TParaFormat;
000056      BEGIN
000057          {$IFC fTrace}BP(10);{$ENDC}
000058          paraFormat := TParaFormat.CREATE(NIL, SELF.Heap, SELF);
000059          SELF.formats.InsLast(paraFormat);
000060          {$IFC fTrace}EP;{$ENDC}
000061      END;
000062
000063
000064  {$S SgTxtCld}
000065  {$IFC fTextTrace}
000066      PROCEDURE TStyleSheet.Fields(PROCEDURE Field(nameAndType: S255));
000067      BEGIN
000068          SUPERSELF.Fields(Field);
000069          Field('formats: TList');
000070          Field('');
000071      END;
000072  {$ENDC}
000073
000074  {$S SgTxtIni}
000075  END; {Methods of TStyleSheet}
000076
000077
000078  METHODS OF TTextRange;
000079
000080  {$S SgTxtHot}
000081      FUNCTION TTextRange.CREATE(object: TObject; heap: THeap;
000082                                          beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
000083                                          endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER): TTextRange;
000084      BEGIN
000085          {$IFC fTrace}BP(6);{$ENDC}
000086          IF object = NIL THEN
000087              object := NewObject(heap, THISCLASS);
000088          SELF := TTextRange(object);
000089          WITH SELF DO
000090              BEGIN
000091              firstPara := beginPara;
```

```
000092              firstIndex := beginIndex;
000093              firstLP := beginLP;
000094              lastPara := endPara;
000095              lastIndex := endIndex;
000096              lastLP := endLP;
000097              END;
000098          {$IFC fTrace}EP;{$ENDC}
000099      END;
000100
000101  {$S SgTxtCld}
000102  {$IFC fTextTrace}
000103      PROCEDURE TTextRange.Fields(PROCEDURE Field(nameAndType: S255));
000104      BEGIN
000105          SUPERSELF.Fields(Field);
000106          Field('firstPara: TEditPara');
000107          Field('firstIndex: LONGINT');
000108          Field('firstLP: INTEGER');
000109          Field('lastPara: TEditPara');
000110          Field('lastIndex: LONGINT');
000111          Field('lastLP: INTEGER');
000112          Field('');
000113      END;
000114  {$ENDC}
000115
000116
000117  {$S SgTxtCld}
000118      PROCEDURE TTextRange.AdjustBy(delta: INTEGER);
000119      BEGIN
000120          {$IFC fTrace}BP(10);{$ENDC}
000121          {$IFC fTrace}EP;{$ENDC}
000122      END;
000123
000124
000125
000126  {$S SgTxtIni}
000127  END;
000128
000129
000130  METHODS OF TText;
000131
000132  {$S SgTxtIni}
000133      FUNCTION TText.CREATE(object: TObject; heap: THeap; itsStyleSheet: TStyleSheet): TText;
000134      VAR aList:          TList;
000135          anotherList:    TList;
000136      BEGIN
000137          {$IFC fTrace}BP(10);{$ENDC}
000138          IF object = NIL THEN
000139              object := NewObject(heap, THISCLASS);
```

```
000140          SELF := TText(object);
000141          aList := TList.CREATE(NIL, heap, 0);
000142          anotherList := TList.CREATE(NIL, heap, 0);
000143          WITH SELF DO
000144              BEGIN
000145              paragraphs := aList;
000146              txtImgList := anotherList;
000147              styleSheet := itsStyleSheet;
000148              END;
000149          {$IFC fTrace}EP;{$ENDC}
000150      END;
000151
000152  {$S SgTxtCld}
000153      PROCEDURE TText.Free;
000154      BEGIN
000155          {$IFC fTrace}BP(10);{$ENDC}
000156          SELF.FreeSelf(TRUE);
000157          {$IFC fTrace}EP;{$ENDC}
000158      END;
000159
000160
000161  {$S SgTxtCld}
000162  {$IFC fTextTrace}
000163      PROCEDURE TText.Fields(PROCEDURE Field(nameAndType: S255));
000164      BEGIN
000165          SUPERSELF.Fields(Field);
000166          Field('paragraphs: TList');
000167          Field('styleSheet: TStyleSheet');
000168          Field('txtImgList: TList');
000169          Field('');
000170      END;
000171  {$ENDC}
000172
000173
000174  {$S SgTxtCld}
000175      PROCEDURE TText.FreeSelf(freeParas: BOOLEAN);
000176      BEGIN
000177          {$IFC fTrace}BP(10);{$ENDC}
000178          SELF.txtImgList.Free;
000179          IF freeParas THEN
000180              Free(SELF.paragraphs)               {Free the paragraphs}
000181          ELSE IF SELF.paragraphs <> NIL THEN           { OR }
000182              SELF.paragraphs.FreeObject;         {Just Free the list}
000183          SUPERSELF.Free;
000184          {$IFC fTrace}EP;{$ENDC}
000185      END;
000186
000187
```

```
000188  {$S SgTxtWrm}
000189      PROCEDURE TText.ChangeSelInOtherPanels(textSelection: TTextSelection);
000190
000191          PROCEDURE ChngPanelSel(obj: TObject);
000192          VAR textImage:  TTextImage;
000193              selection:  TSelection;
000194              panel:      TPanel;
000195          BEGIN
000196              {$IFC fTrace}BP(10);{$ENDC}
000197              textImage := TTextImage(obj);
000198              panel := textImage.view.panel;
000199              selection := panel.selection;
000200              {We only unhighlight and replace the last non-NIL coSelection.  In most cases, where
000201               there is no coSelection, we unhighlight and replace the panel selection and
000202               everything is hunky-dory}
000203              WHILE selection.coSelection <> NIL DO
000204                  selection := selection.coSelection;
000205
000206              {Don't change selection in same panel}
000207              IF selection.panel <> textSelection.panel THEN
000208                  selection := selection.FreedAndReplacedBy(textSelection.ReplicateForOtherPanel(textImage));
000209              {$IFC fTrace}EP;{$ENDC}
000210          END;
000211
000212      BEGIN
000213          {$IFC fTrace}BP(10);{$ENDC}
000214          IF SELF.txtImgList.size > 1 THEN
000215              SELF.txtImgList.Each(ChngPanelSel);
000216          {$IFC fTrace}EP;{$ENDC}
000217      END;
000218
000219
000220  {$S SgTxtCld}
000221      PROCEDURE TText.DelPara(delPara: TEditPara;  fFree: BOOLEAN);
000222      VAR (*
000223          i:          INTEGER;
000224          numImages:  INTEGER;
000225          *)
000226          s:          TListScanner;
000227          paraImage:  TParaImage;
000228      BEGIN
000229          {$IFC fTrace}BP(10);{$ENDC}
000230          (*
000231          numImages := delPara.NumImages;
000232          FOR i := 1 TO numImages DO
000233          *)
000234          s := delPara.images.Scanner;
000235          WHILE s.Scan(paraImage) DO
```

```
000236              BEGIN
000237              (*
000238              paraImage := delPara.images[1];
000239              *)
000240              paraImage.textImage.imageList.DelObject(paraImage, FALSE);
000241              s.Delete(FALSE);
000242              paraImage.Free;
000243              END;
000244          {NOTE: We do not delete the paragraph from our own paragraphs list because this is usually
000245                 called while scanning that list and we would screw up its scanner if we removed the
000246                 paragraph from the list}
000247          IF fFree THEN
000248              delPara.free;
000249          {$IFC fTrace}EP;{$ENDC}
000250      END;
000251
000252
000253  {$S SgTxtCld}
000254      PROCEDURE TText.Draw;
000255          PROCEDURE DrawInImage(obj: TObject);
000256          VAR textImage:  TTextImage;
000257              PROCEDURE DrawOnPad;
000258              BEGIN
000259                  textImage.Draw;
000260              END;
000261          BEGIN
000262              textImage := TTextImage(obj);
000263              textImage.view.panel.OnAllPadsDo(DrawOnPad)
000264          END;
000265      BEGIN
000266          {$IFC fTrace}BP(10);{$ENDC}
000267          SELF.txtImgList.Each(DrawInImage);
000268          {$IFC fTrace}EP;{$ENDC}
000269      END;
000270
000271
000272  {$S SgTxtHot}
000273      PROCEDURE TText.HiliteRange(highTransit: THighTransit; textRange: TTextRange; wholePara: BOOLEAN);
000274      BEGIN
000275          {$IFC fTrace}BP(10);{$ENDC}
000276          WITH textRange DO
000277              {$H-}
000278              SELF.HiliteParagraphs(highTransit, firstIndex, firstLP, lastIndex, lastLP, wholePara);
000279              {$H+}
000280          {$IFC fTrace}EP;{$ENDC}
000281      END;
000282
000283
```

```
000284  {$S SgTxtHot}
000285      { TText.HiliteParagraphs tells each text image to hilite its panel's text selection on each pad.  It
000286        calls TTextImage.HiliteText which assumes we are already focussed on a pad }
000287      PROCEDURE TText.HiliteParagraphs(highTransit: THighTransit;
000288                                                  startIndex: LONGINT; startLP: INTEGER;
000289                                                  endIndex: LONGINT; endLP: INTEGER; wholePara: BOOLEAN);
000290          PROCEDURE HiliteInImage(obj: TObject);
000291          VAR selection:  TSelection;
000292              textImage:  TTextImage;
000293
000294              PROCEDURE HiliteOnPad;
000295              BEGIN
000296                  textImage.HiliteText(highTransit, startIndex, startLP, endIndex, endLP, wholePara);
000297              END;
000298          BEGIN
000299              selection := TTextImage(obj).view.panel.selection;
000300              WHILE selection.coSelection <> NIL DO
000301                  selection := selection.coSelection;
000302              textImage := TTextSelection(selection).textImage;
000303              textImage.view.panel.OnAllPadsDo(HiliteOnPad);
000304          END;
000305      BEGIN
000306          {$IFC fTrace}BP(10);{$ENDC}
000307          SELF.txtImgList.Each(HiliteInImage);
000308          {$IFC fTrace}EP;{$ENDC}
000309      END;
000310
000311
000312  {$S SgTxtIni}
000313      FUNCTION TText.DfltTextImage(view: TView; imageLRect: LRect; imgIsGrowable: BOOLEAN): TTextImage;
000314      VAR textImage:  TTextImage;
000315      BEGIN
000316          {$IFC fTrace}BP(10);{$ENDC}
000317          textImage := TTextImage.CREATE(NIL, SELF.Heap, view, imageLRect, SELF, imgIsGrowable);
000318          SELF.txtImgList.InsLast(textImage);
000319          SELF.paragraphs.InsLast(textImage.NewEditPara(0, TParaFormat(SELF.styleSheet.formats.First)));
000320          SELF.RecomputeImages;
000321          DfltTextImage := textImage;
000322          {$IFC fTrace}EP;{$ENDC}
000323      END;
000324
000325
000326  {$S SgTxtCld}
000327      PROCEDURE TText.InsParaAfter(existingPara: TEditPara;  newPara: TEditPara);
000328
000329          PROCEDURE InsertPara(obj: TObject);
000330          BEGIN
000331              TTextImage(obj).InsertNewPara(existingPara, newPara);
```

```
000332          END;
000333      BEGIN
000334          {$IFC fTrace}BP(10);{$ENDC}
000335          SELF.txtImgList.Each(InsertPara);
000336          SELF.paragraphs.InsAt(SELF.paragraphs.Pos(0, existingPara) + 1, newPara);
000337          {$IFC fTrace}EP;{$ENDC}
000338      END;
000339
000340
000341  {$S SgTxtCld}
000342      PROCEDURE TText.Invalidate;
000343
000344          PROCEDURE InvalInImage(obj: TObject);
000345          VAR textImage:  TTextImage;
000346              PROCEDURE InvalOnPad;
000347              BEGIN
000348                  textImage.Invalidate;
000349              END;
000350          BEGIN
000351              textImage := TTextImage(obj);
000352              IF textImage.imageList.Size > 0 THEN
000353                  textImage.view.panel.OnAllPadsDo(InvalOnPad);
000354          END;
000355      BEGIN
000356          {$IFC fTrace}BP(10);{$ENDC}
000357          SELF.txtImgList.Each(InvalInImage);
000358          {$IFC fTrace}EP;{$ENDC}
000359      END;
000360
000361
000362  {$S SgTxtHot}
000363      PROCEDURE TText.MarkChanged(textRange: TTextRange);
000364
000365          PROCEDURE Mark(obj: TObject);
000366          VAR textImage:  TTextImage;
000367              selection:  TSelection;
000368          BEGIN
000369              selection := TTextImage(obj).view.panel.selection;
000370              WHILE selection.coSelection <> NIL DO
000371                  selection := selection.coSelection;
000372              textImage := TTextSelection(selection).textImage;
000373              WITH textRange DO
000374                  {$H-}
000375                  textImage.MarkChanged(firstIndex, firstLP, lastIndex, lastLP);
000376                  {$H+}
000377          END;
000378      BEGIN
000379          {$IFC fTrace}BP(10);{$ENDC}
```

```
000380          SELF.txtImgList.Each(Mark);
000381          {$IFC fTrace}EP;{$ENDC}
000382      END;
000383
000384
000385  {$S SgTxtHot}
000386      PROCEDURE TText.RecomputeImages;
000387          PROCEDURE ReImage(obj: TObject);
000388          VAR textImage:  TTextImage;
000389              padCount:   INTEGER;
000390              numPads:    INTEGER;
000391
000392              PROCEDURE ImageOnPad;
000393              BEGIN
000394                  padCount := padCount+1;
000395                  {The first parameter in textImage.RecomputeImages says we want to draw, but it will
000396                   call view.OKToDrawIn to be sure its Okay.
000397                   The last parameter is TRUE when we are drawing on the last pad.  RecomputeImages
000398                   and DrawOrInval will then set the valid bits on the images to TRUE}
000399                  IF padCount = 1 THEN
000400                      textImage.RecomputeImages(actionDraw, (numPads = 1))
000401                  ELSE
000402                      textImage.DrawOrInval(padCount = numPads);
000403              END;
000404          BEGIN
000405              textImage := TTextImage(obj);
000406              numPads := textImage.view.panel.panes.size;
000407              padCount := 0;
000408              textImage.view.panel.OnAllPadsDo(ImageOnPad);
000409          END;
000410      BEGIN
000411          {$IFC fTrace}BP(10);{$ENDC}
000412          SELF.txtImgList.Each(ReImage);
000413          {$IFC fTrace}EP;{$ENDC}
000414      END;
000415
000416
000417  {$S SgTxtWrm}
000418      FUNCTION TText.SelectAll(textImage: TTextImage): TTextSelection;
000419      VAR lastPara:   TEditPara;
000420          textSelection:      TTextSelection;
000421      BEGIN
000422          {$IFC fTrace}BP(10);{$ENDC}
000423          lastPara := TEditPara(SELF.paragraphs.Last);
000424          textSelection := textImage.NewTextSelection(TEditPara(SELF.paragraphs.First), 1, 0,
000425                                                      lastPara, SELF.paragraphs.Size, lastPara.size);
000426          SelectAll := textSelection;
000427          {$IFC fTrace}EP;{$ENDC}
```

```
000428      END;
000429
000430  {$S SgTxtIni}
000431  BEGIN
000432      fTextTrace := FALSE;
000433  END; {Methods of TText}
000434
000435
000436  METHODS OF TTextImage;
000437
000438  {$S SgTxtIni}
000439      FUNCTION TTextImage.CREATE(object: TObject; heap: THeap; itsView: TView;
000440                                      itsLRect: LRect; itsText: TText; isGrowable: BOOLEAN): TTextImage;
000441      VAR imgList:    TList;
000442      BEGIN
000443          {$IFC fTrace}BP(10);{$ENDC}
000444          IF object = NIL THEN
000445              object := NewObject(heap, THISCLASS);
000446          SELF := TTextImage(TImage.CREATE(object, heap, itsLRect, itsView));
000447          imgList := TList.CREATE(NIL, heap, 0);
000448
000449          WITH SELF DO
000450              BEGIN
000451              text := itsText;
000452              imageList := imgList;
000453              tickCount := 0;
000454              growsDynamically := isGrowable;
000455              minHeight := itsLRect.bottom - itsLRect.top;
000456              formerBottom := itsLRect.top;
000457              updateLRect := zeroLRect;
000458              firstLinePixel := 0;
000459              useFirstPixel := FALSE;
000460              firstIndex := 1;
000461              startLP := 0;
000462              endLP := 0;
000463              {app must set these properly if using multiple linked text images}
000464              prevTxtImg := NIL;
000465              nextTxtImg := NIL;
000466              headTxtImg := SELF;
000467              tailTxtImg := SELF;
000468              END;
000469          {$IFC fTrace}EP;{$ENDC}
000470      END;
000471
000472  {$S SgTxtCld}
000473      {This frees all text images and their paraImages in the text image chain.
000474       It does NOT free any paragraphs, text objects, or paraFormats.  Call this only once
000475       for each text image chain (NOT for each text image in the chain}
```

```
000476      PROCEDURE TTextImage.Free;
000477      VAR textImage:  TTextImage;
000478          next:       TTextImage;
000479      BEGIN
000480          {$IFC fTrace}BP(10);{$ENDC}
000481          IF SELF.headTxtImg = SELF THEN
000482              {Think about freeing text here if this is its only text image, but beware of circular frees}
000483                  BEGIN
000484                  textImage := SELF;
000485                  WHILE textImage <> NIL DO
000486                      BEGIN
000487                      textImage.imageList.Free;
000488                      next := textImage.nextTxtImg;
000489                      textImage.FreeObject;
000490                      textImage := next;
000491                      END;
000492                  END
000493          ELSE
000494              SELF.headTxtImg.Free;
000495          {$IFC fTrace}EP;{$ENDC}
000496      END;
000497
000498
000499  {$S SgTxtCld}
000500      {Frees just one text image in the chain; pays no attention to links}
000501      PROCEDURE TTextImage.FreeOneTextImage;
000502      VAR textImage:  TTextImage;
000503          next:       TTextImage;
000504      BEGIN
000505          {$IFC fTrace}BP(10);{$ENDC}
000506          SELF.imageList.Free;
000507          SELF.FreeObject;
000508          {$IFC fTrace}EP;{$ENDC}
000509      END;
000510
000511
000512
000513  {$S SgTxtCld}
000514  {$IFC fTextTrace}
000515      PROCEDURE TTextImage.Fields(PROCEDURE Field(nameAndType: S255));
000516      BEGIN
000517          SUPERSELF.Fields(Field);
000518          Field('text: TText');
000519          Field('imageList: TList');
000520          Field('tickCount: INTEGER');
000521          Field('growsDynamically: BOOLEAN');
000522          Field('minHeight: INTEGER');
000523          Field('formerBottom: LONGINT');
```

```
000524          Field('updateLRect: LRect');
000525          Field('firstLinePixel: LONGINT');
000526          Field('useFirstPixel: BOOLEAN');
000527          Field('firstIndex: LONGINT');
000528          Field('startLP: INTEGER');
000529          Field('endLP: INTEGER');
000530          Field('prevTxtImg: TTextImage');
000531          Field('nextTxtImg: TTextImage');
000532          Field('headTxtImg: TTextImage');
000533          Field('tailTxtImg: TTextImage');
000534          Field('');
000535      END;
000536  {$ENDC}
000537
000538
000539  {$S SgTxtCld}
000540      PROCEDURE TTextImage.AddImage(paraImage: TParaImage);
000541      BEGIN
000542          {$IFC fTrace}BP(10);{$ENDC}
000543          SELF.imageList.InsLast(paraImage);
000544          {$IFC fTrace}EP;{$ENDC}
000545      END;
000546
000547
000548  {$S SgTxtCld}
000549      PROCEDURE TTextImage.DelImagesWith(delPara: TEditPara);
000550      VAR (*
000551          i,j:        INTEGER;
000552          numImages:  INTEGER;
000553          *)
000554          s:          TListScanner;
000555          paraImage:  TParaImage;
000556      BEGIN
000557          {$IFC fTrace}BP(10);{$ENDC}
000558          s := delPara.images.Scanner;
000559          WHILE s.Scan(paraImage) DO
000560              IF paraImage.textImage.headTxtImg = SELF.headTxtImg THEN
000561                  BEGIN
000562                  paraImage.textImage.imageList.DelObject(paraImage, FALSE);
000563                  s.Delete(FALSE);
000564                  paraImage.Free;
000565                  END;
000566          (*
000567          numImages := delPara.NumImages;
000568          j := 1;
000569          FOR i := 1 TO numImages DO
000570              BEGIN
000571              {paraImage.Free calls paragraph.DelImage which shifts rest
```

```
000572              of images left so next image will always be at position j}
000573             paraImage := delPara.images[j];
000574             IF paraImage.textImage.headTxtImg = SELF.headTxtImg THEN
000575                 paraImage.textImage.imageList.DelObject(paraImage, TRUE)
000576             ELSE
000577                 j := j + 1;
000578             END;
000579         *)
000580         {$IFC fTrace}EP;{$ENDC}
000581     END;
000582
000583
000584 {$S SgTxtWrm}
000585     PROCEDURE TTextImage.Draw;
000586         PROCEDURE ReDraw(obj: TObject);
000587         BEGIN
000588             IF LRectIsVisible(TParaImage(obj).extentLRect) THEN
000589                 TParaImage(obj).RedrawLines(0, MAXINT);
000590         END;
000591     BEGIN
000592         {$IFC fTrace}BP(10);{$ENDC}
000593         PicGrpBegin;
000594         SELF.imageList.Each(ReDraw);
000595         {Now tell the next textImage in the chain to draw itself}
000596         IF SELF.nextTxtImg <> NIL THEN
000597             SELF.nextTxtImg.Draw;
000598
000599         PicGrpEnd;
000600         {$IFC fTrace}EP;{$ENDC}
000601     END; {Draw}
000602
000603
000604 {$S SgTxtCld}
000605     PROCEDURE TTextImage.DrawImages(fDraw: BOOLEAN);
000606         PROCEDURE ReDraw(obj: TObject);
000607         BEGIN
000608             IF LRectIsVisible(TParaImage(obj).extentLRect) THEN
000609                 TParaImage(obj).RedrawLines(0, MAXINT);
000610         END;
000611     BEGIN
000612         {$IFC fTrace}BP(10);{$ENDC}
000613         SELF.imageList.Each(ReDraw);
000614         {Now tell the next textImage in the chain to draw itself}
000615         IF SELF.nextTxtImg <> NIL THEN
000616             SELF.nextTxtImg.DrawImages(fDraw);
000617         {$IFC fTrace}EP;{$ENDC}
000618     END; {DrawImages}
000619
```

```
000620
000621  {$S SgTxtCld}
000622      PROCEDURE TTextImage.DrawOrInval(invalBits: BOOLEAN);
000623      VAR fDraw:      BOOLEAN;
000624          r:          LRect;
000625
000626          PROCEDURE DrawFiltImage(obj: TObject);
000627          VAR paraImage:  TParaImage;
000628
000629              PROCEDURE DrawImage(obj: TObject);
000630              VAR leftPixel:  LONGINT;
000631                  rightPixel: LONGINT;
000632                  styleIndex: INTEGER;
000633
000634                  PROCEDURE DrawLine(obj: TObject);
000635                  VAR lineInfo:   TLineInfo;
000636                      dummy:      INTEGER;
000637                  BEGIN
000638                      lineInfo := TLineInfo(obj);
000639                      IF NOT lineInfo.valid THEN
000640                          BEGIN
000641                          r := lineInfo.lineLRect;
000642                          r.left := lineInfo.leftCoord(leftPixel);
000643                          r.right := lineInfo.leftCoord(rightPixel);
000644                          IF fDraw THEN
000645                              BEGIN
000646                              FillLRect(r, lPatWhite);
000647                              IF lineInfo.startLP <> lineInfo.lastDrawnLP THEN
000648                                  BEGIN
000649                                  MoveToL(lineInfo.lineLRect.left, lineInfo.lineLRect.top+lineInfo.lineAscent);
000650                                  paraImage.FastDrawLine(lineInfo.startLP, lineInfo.lastDrawnLP, TRUE,
000651                                                         FALSE, dummy, styleIndex);
000652                                  END;
000653                              END
000654                          ELSE
000655                              thePad.InvalLRect(r);
000656                          lineInfo.valid := invalBits;
000657                          END;
000658                  END;
000659
000660              BEGIN
000661                  IF paraImage.wasOffset THEN
000662                      BEGIN
000663                      {possibly use ScrollRect here later?}
000664                      r := paraImage.extentLRect;
000665                      InsetLRect(r, -1, 0);
000666                      IF fDraw THEN
000667                          BEGIN
```

```
000668                          FillLRect(r, lPatWhite);
000669                          paraImage.RedrawLines(0, MAXINT);
000670                          END
000671                      ELSE
000672                          thePad.InvalLRect(r);
000673                      paraImage.wasOffset := NOT invalBits;
000674                      END
000675                  ELSE IF paraImage.changed THEN
000676                      BEGIN
000677                      leftPixel := paraImage.extentLRect.left-1;
000678                      rightPixel := paraImage.extentLRect.right+1;
000679                      styleIndex := 1;
000680                      paraImage.lineList.Each(DrawLine);
000681                      paraImage.changed := NOT invalBits;
000682                      END;
000683              END;
000684
000685          BEGIN
000686              paraImage := TParaImage(obj);
000687              SELF.FilterAndDo(paraImage, DrawImage);
000688          END;
000689
000690      BEGIN
000691          {$IFC fTrace}BP(10);{$ENDC}
000692          fDraw := SELF.view.OKToDrawIn(SELF.extentLRect) AND NOT deferUpdate;
000693          SELF.imageList.Each(DrawFiltImage);
000694
000695          IF NOT EmptyLRect(SELF.updateLRect) THEN
000696              BEGIN
000697              IF fDraw THEN
000698                  FillLRect(SELF.updateLRect, lPatWhite)
000699              ELSE
000700                  thePad.InvalLRect(SELF.updateLRect);
000701              IF invalBits THEN
000702                  SELF.updateLRect := zeroLRect;
000703              END;
000704          {Now tell the next textImage in the chain to draw itself}
000705          IF SELF.nextTxtImg <> NIL THEN
000706              SELF.nextTxtImg.DrawOrInval(invalBits);
000707          {$IFC fTrace}EP;{$ENDC}
000708      END;
000709
000710
000711  {$S SgTxtHot}
000712      PROCEDURE TTextImage.FindParaAndLp(lPt: LPoint; VAR paraImage: TParaImage;
000713                                                          VAR paraIndex: LONGINT; VAR aLP: INTEGER);
000714      VAR distanceDown:   INTEGER;
000715          s:              TListScanner;
```

```
000716          {$IFC fTextTrace}
000717          str:            STR255;
000718          {$ENDC}
000719      BEGIN
000720          {$IFC fTrace}BP(9);{$ENDC}
000721
000722          {It is assumed that the caller of this routine has already determined that lPt is in this
000723           textImage, so we will not check nextTextImg if the image list is exhausted}
000724          distanceDown := SELF.extentLRect.top;
000725          s := SELF.imageList.Scanner;
000726          WHILE s.Scan(paraImage) DO
000727              BEGIN
000728              distanceDown := distanceDown + paraImage.height;
000729              paraIndex := s.Position;
000730              IF lPt.v <= distanceDown THEN
000731                  s.Done;
000732              END;
000733
000734          paraIndex := paraIndex + SELF.firstIndex - 1;
000735
000736          IF lPt.v > distanceDown THEN
000737              paraImage := TParaImage(SELF.imageList.Last);
000738
000739          aLP := paraImage.LpWithLPt(lPt);
000740          {$IFC fTextTrace}
000741          IF fTextTrace THEN
000742              BEGIN
000743              LIntToHex(ORD(paraImage), @str);
000744              writeln('*** End FindParaAndLp: lPt= (',lPt.v:4, ',', lPt.h:4, ');  paraImage, index, lp = (',
000745                                      str, ',', paraIndex:1, ',', aLP:3, ')');
000746              END;
000747          {$ENDC}
000748          {$IFC fTrace}EP;{$ENDC}
000749      END;
000750
000751
000752
000753  {$S SgTxtHot}
000754      FUNCTION  TTextImage.FindTextImage(VAR mouseLPt: LPoint; VAR firstTxtImg: TTextImage): TTextImage;
000755      VAR textImage:      TTextImage;
000756          stillLooking:   BOOLEAN;
000757          foundIt:        BOOLEAN;
000758      BEGIN
000759          {$IFC fTrace}BP(9);{$ENDC}
000760          {This looks around for a textImage that contains the mouseLPt and has some images.  If it
000761           finds a textImage that contains the point but does not have any images, then it
000762           returns the first previous textImage that does have some images and changes mouseLPt to be
000763           the bottom right point of that textImage.
```

```
000764          If it doesn't find any textImages that contain the point it returns the first previous
000765          textImage that does have some images.  Also, it returns which of the textImages
000766          (SELF or the found one) that comes first in the textImage chain}
000767
000768          firstTxtImg := SELF;
000769          {Start with most common case and then try others}
000770          IF (SELF.imageList.Size > 0) AND
000771                  (LPtInLRect(mouseLPt, SELF.extentLRect) OR SELF.growsDynamically) THEN
000772              FindTextImage := SELF
000773          ELSE
000774              BEGIN
000775              textImage := SELF;
000776              stillLooking := TRUE;
000777              foundIt := FALSE;
000778              WHILE stillLooking DO
000779                  BEGIN
000780                  {First look in following boxes}
000781                  IF LPtInLRect(mouseLPt, textImage.extentLRect) THEN
000782                      BEGIN
000783                      {if box found but no images in it, then link back}
000784                      WHILE (textImage <> textImage.headTxtImg) AND (textImage.imageList.Size = 0) DO
000785                          BEGIN
000786                          textImage := textImage.prevTxtImg;
000787                          mouseLPt := textImage.extentLRect.botRight;
000788                          END;
000789                      foundIt := TRUE;
000790                      stillLooking := FALSE;
000791                      END
000792                  ELSE IF textImage.nextTxtImg <> NIL THEN
000793                      textImage := textImage.nextTxtImg
000794                  ELSE stillLooking := FALSE;
000795                  END;
000796
000797              IF foundIt THEN
000798                  FindTextImage := textImage
000799              ELSE
000800                  {Still didn't find it?  Look in previous boxes}
000801                  BEGIN
000802                  stillLooking := TRUE;
000803                  WHILE stillLooking DO
000804                      BEGIN
000805                      IF LPtInLRect(mouseLPt, textImage.extentLRect) THEN
000806                          BEGIN
000807                          WHILE (textImage <> textImage.headTxtImg) AND (textImage.imageList.Size = 0) DO
000808                              BEGIN
000809                              textImage := textImage.prevTxtImg;
000810                              mouseLPt := textImage.extentLRect.botRight;
000811                              END;
```

```
000812                          foundIt := TRUE;
000813                          stillLooking := FALSE;
000814                          END
000815                      ELSE IF textImage.prevTxtImg <> NIL THEN
000816                          textImage := textImage.prevTxtImg
000817                      ELSE stillLooking := FALSE;
000818                      END;
000819                  IF foundIt THEN
000820                      BEGIN
000821                      FindTextImage := textImage;
000822                      firstTxtImage := textImage;
000823                      END
000824                  ELSE
000825                      BEGIN
000826                      {mouseLPt didn't fall in any of the text images, so return SELF or the first previous
000827                       text image that has a paraimage}
000828                      textImage := SELF;
000829                      WHILE (textImage <> textImage.headTxtImg) AND (textImage.imageList.Size = 0) DO
000830                         BEGIN
000831                         textImage := textImage.prevTxtImg;
000832                         mouseLPt := textImage.extentLRect.botRight;
000833                         END;
000834                      FindTextImage := textImage;
000835                      END;
000836                  END;
000837              END;
000838          {$IFC fTrace}EP;{$ENDC}
000839      END;
000840
000841
000842  {$S SgTxtHot}
000843      PROCEDURE TTextImage.GetImageRange(firstIndex: LONGINT; VAR firstLP: INTEGER;
000844                                         lastIndex:  LONGINT; VAR lastLP:  INTEGER;
000845                                         VAR firstImage, lastImage: TParaImage);
000846
000847      {Diagram of input vs output:  --- = characters not displayed by this textImage chain;
000848                                    xxx = characters that are displayed by this textImage chain;
000849
000850          -----------xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx----------
000851            ^    ^    ^       ^                     ^    ^  ^
000852            1    2    3       4                     5    6  7
000853
000854
000855          input   imageWith     output
000856           1,2        N,N         N,N
000857           1,3        N,3         N,3
000858           1,4        N,4         3,4
000859           1,5        N,5         3,5
```

```
000860          1,6        N,N         3,5
000861          3,3        3,3         3,3             N = NIL
000862          3,4        3,4         3,4
000863          4,5        4,5         4,5
000864          4,6        4,N         4,5
000865          5,5        5,5         5,5
000866          5,6        5,N         5,N
000867          6,7        N,N         N,N
000868       }
000869
000870         FUNCTION GetFirstOrLast(index: LONGINT; VAR lp: INTEGER): TParaImage;
000871         VAR paraImage:  TParaImage;
000872             lastTxtImg: TTextImage;
000873         BEGIN
000874             {$IFC fTrace}BP(10);{$ENDC}
000875             IF (index < SELF.headTxtImg.firstIndex) OR
000876                ((index = SELF.headTxtImg.firstIndex) AND (lp < SELF.headTxtImg.startLP)) THEN
000877                 BEGIN
000878                 paraImage := TParaImage(SELF.headTxtImg.imageList.First);
000879                 lp := paraImage.startLP;
000880                 END
000881             ELSE
000882                 BEGIN
000883                 lastTxtImg := SELF.tailTxtImg;
000884                 WHILE lastTxtImg.imageList.Size <= 0 DO
000885                     lastTxtImg := lastTxtImg.prevTxtImg;
000886                 paraImage := TParaImage(lastTxtImg.imageList.Last);
000887                 lp := paraImage.endLP;
000888                 END;
000889             GetFirstOrLast := paraImage;
000890             {$IFC fTrace}EP;{$ENDC}
000891         END;
000892      BEGIN
000893         {$IFC fTrace}BP(10);{$ENDC}
000894         firstImage := SELF.ImageWith(TEditPara(SELF.text.paragraphs.At(firstIndex)), firstLP);
000895         lastImage := SELF.ImageWith(TEditPara(SELF.text.paragraphs.At(lastIndex)), lastLP);
000896
000897         IF (firstImage = NIL) OR (lastImage = NIL) THEN
000898             BEGIN
000899             IF firstImage = NIL THEN
000900                 IF lastImage <> NIL THEN
000901                     BEGIN
000902                     firstImage := GetFirstOrLast(firstIndex, firstLP);
000903                     IF (firstImage = lastImage) THEN
000904                         IF (firstLP = lastLP) THEN
000905                             firstImage := NIL;
000906                     END
000907                 ELSE
```

```
000908                      BEGIN
000909                      firstImage := GetFirstOrLast(firstIndex, firstLP);
000910                      lastImage := GetFirstOrLast(lastIndex, lastLP);
000911                      IF (firstImage = lastImage) AND (firstLP = lastLP) THEN
000912                          BEGIN
000913                          firstImage := NIL;
000914                          lastImage := NIL;
000915                          END;
000916                      END
000917                ELSE
000918                    BEGIN
000919                    lastImage := GetFirstOrLast(lastIndex, lastLP);
000920                    IF (firstImage = lastImage) THEN
000921                        IF (firstLP = lastLP) THEN
000922                            lastImage := NIL;
000923                    END;
000924                END;
000925            {$IFC fTrace}EP;{$ENDC}
000926        END;
000927
000928
000929  {$S SgTxtHot}
000930        PROCEDURE TTextImage.HiliteText(highTransit: THighTransit;
000931                                                    startIndex: LONGINT; startLP: INTEGER;
000932                                                    endIndex: LONGINT; endLP: INTEGER; wholePara: BOOLEAN);
000933        LABEL 1;
000934        VAR startImage:     TParaImage;
000935            endImage:       TParaImage;
000936            r:              LRect;
000937            lineInfo:       TLineInfo;
000938            paraImage:      TParaImage;
000939            sImg, sViewSt:  TListScanner;
000940            startLine:      INTEGER;
000941            endLine:        INTEGER;
000942            startPixel:     LONGINT;
000943            endPixel:       LONGINT;
000944            lMargPixel:     LONGINT;
000945            rMargPixel:     LONGINT;
000946            i:              INTEGER;
000947            textImage:      TTextImage;
000948            stillOkay:      BOOLEAN;
000949            {$IFC fTextTrace}
000950            str1:           STR255;     {for debug output}
000951            str2:           STR255;     {for debug output}
000952            {$ENDC}
000953        BEGIN
000954            {$IFC fTrace}BP(10);{$ENDC}
000955            {$IFC fTextTrace}
```

```
000956          IF fTextTrace THEN
000957              BEGIN
000958              Writeln('*** In HiliteText:  [(', startIndex:1, ',', startLP:1, ') , (',
000959                                          endIndex:1, ',', endLP:1,
000960                                          ')];  highTransit = ', ORD(highTransit):2);
000961              LIntToHex(ORD(SELF), @str1);
000962              WriteLn('*** SELF = ', str1, '  SELF.endLP=', SELF.endLP:1);
000963              END;
000964          {$ENDC}
000965
000966          IF (startIndex = endIndex) AND (startLP = endLP) THEN
000967              BEGIN
000968              startImage := SELF.ImageWith(TEditPara(SELF.text.paragraphs.At(startIndex)), startLP);
000969              endImage := startImage;
000970              stillOkay := startImage <> NIL;
000971              END
000972          ELSE
000973              BEGIN
000974              SELF.GetImageRange(startIndex, startLP, endIndex, endLP, startImage, endImage);
000975              stillOkay := (startImage <> NIL) AND (endImage <> NIL);
000976              END;
000977
000978          IF stillOkay THEN
000979              BEGIN
000980              textImage := startImage.textImage;
000981
000982              lMargPixel := textImage.extentLRect.left - 1;
000983              rMargPixel := textImage.extentLRect.right + 1;
000984
000985              IF highTransit = hDimToOn THEN
000986                  BEGIN
000987                  SELF.HiliteText(hDimToOff, startIndex, startLP, endIndex, endLP, wholePara);
000988                  highTransit := hOffToOn;
000989                  END
000990              ELSE IF highTransit = hOffToDim THEN
000991                  BEGIN
000992                  SELF.HiliteText(hOffToOn, startIndex, startLP, endIndex, endLP, wholePara);
000993                  highTransit := hOnToDim;
000994                  END;
000995
000996              IF highTransit <> hNone THEN
000997                  BEGIN
000998                  SetPenState(highPen[highTransit]);
000999                  IF highTransit = hOnToDim THEN
001000                      PenMode(notPatBic); { hOnToDim  => change background from black to gray }
001001
001002                  sImg := textImage.imageList.Scanner;
001003                  WHILE sImg.Scan(paraImage) DO
```

```
001004                      IF paraImage = startImage THEN
001005                          GOTO 1;
001006
001007                  1:
001008                  paraImage.LocateLP(startLP, startLine, startPixel);
001009
001010                  sViewSt := paraImage.lineList.Scanner;
001011                  i := 0;
001012                  REPEAT
001013                      IF sViewSt.Scan(lineInfo) THEN
001014                          i := i+1;
001015                  UNTIL i = startLine;
001016
001017                  r := lineInfo.lineLRect;
001018                  IF wholePara THEN
001019                      r.left := lineInfo.LeftCoord(lMargPixel)
001020                  ELSE
001021                      r.left := startPixel;
001022                  r.right := lineInfo.RightCoord(rMargPixel);
001023                  WHILE paraImage <> endImage DO
001024                      BEGIN
001025                      WHILE sViewSt.Scan(lineInfo) DO
001026                          BEGIN
001027                          PaintLRect(r);
001028                          {$IFC fTextTrace}
001029                          IF fTextTrace THEN
001030                              BEGIN
001031                              WriteLn('*** Within HiliteText: about to paintLRect: [(', r.left:1, ',',
001032                                                  r.top:1,'),(', r.left:1,',',r.right:1,')]' );
001033                              LIntToHex(ORD(paraImage), @str1);
001034                              WriteLn('*** current paraImage=',str1);
001035                              END;
001036                          {$ENDC}
001037                          r := lineInfo.lineLRect;
001038                          r.left := lineInfo.LeftCoord(lMargPixel);
001039                          r.right := lineInfo.RightCoord(rMargPixel);
001040                          END;
001041                      IF sImg.Scan(paraImage) THEN
001042                          sViewSt := paraImage.lineList.Scanner
001043                      ELSE
001044                          BEGIN
001045                          textImage := textImage.nextTxtImage;
001046                          lMargPixel := textImage.extentLRect.left-1;
001047                          rMargPixel := textImage.extentLRect.right+1;
001048                          sImg := textImage.imageList.Scanner;
001049                          IF sImg.Scan(paraImage) THEN
001050                              sViewSt := paraImage.lineList.Scanner;
001051                          END;
```

```
001052                  END;
001053              paraImage.LocateLP(endLP, endLine, endPixel);
001054
001055              IF startImage <> endImage THEN
001056                  BEGIN
001057                  PaintLRect(r);
001058                  sViewSt := paraImage.lineList.Scanner;
001059                  IF sViewSt.Scan(lineInfo) THEN
001060                      BEGIN
001061                      r := lineInfo.lineLRect;
001062                      r.left := lineInfo.LeftCoord(lMargPixel);
001063                      r.right := lineInfo.RightCoord(rMargPixel);
001064                      END;
001065                  i := 1;
001066                  END;
001067
001068              WHILE i <> endLine DO
001069                  BEGIN
001070                  PaintLRect(r);
001071                  IF sViewSt.Scan(lineInfo) THEN
001072                      BEGIN
001073                      r := lineInfo.lineLRect;
001074                      r.left := lineInfo.LeftCoord(lMargPixel);
001075                      r.right := lineInfo.RightCoord(rMargPixel);
001076                      END;
001077                  i := i+1;
001078                  END;
001079
001080              IF wholePara THEN
001081                  r.right := lineInfo.RightCoord(rMargPixel)
001082              ELSE
001083                  BEGIN
001084                  r.right := endPixel;
001085                  {Add extra pixel if this is insertion point}
001086                  IF (startImage = endImage) AND (startLP = endLP) THEN
001087                      r.right := r.right + 1;
001088                  END;
001089              PaintLRect(r);
001090
001091              Free(sViewSt);
001092              Free(sImg);
001093
001094              IF highTransit = hDimToOff THEN { XORing out gray leaves holes in chars }
001095                  BEGIN
001096                  {later, we'll minimize this, if necessary}
001097                  SELF.Draw;     { so redraw characters }
001098                  END;
001099              END;
```

```
001100              END
001101          ELSE
001102              BEGIN
001103              {$IFC fTextTrace}
001104              IF fTextTrace THEN
001105                  BEGIN
001106                  Writeln('*** In HiliteText:  Images NIL (DID NOT HILITE))');
001107                  END;
001108              {$ENDC}
001109              END;
001110          {$IFC fTrace}EP;{$ENDC}
001111      END; {HiliteText}
001112
001113
001114  {$S SgTxtHot}
001115      {Given a paragraph and lp finds the paraImage that displays it in this textImage chain.
001116       Returns NIL if not found.}
001117      FUNCTION  TTextImage.ImageWith(paragraph: TEditPara; lp: INTEGER): TParaImage;
001118      VAR paraImage:      TParaImage;
001119          altParaImage:   TParaImage;
001120          s:              TListScanner;
001121          (*
001122          i:          INTEGER;
001123          *)
001124          {$IFC fTextTrace}
001125          str:         STR255;
001126          {$ENDC}
001127      BEGIN
001128          {$IFC fTrace}BP(9);{$ENDC}
001129          {$IFC fTextTrace}
001130          IF fTextTrace THEN
001131              BEGIN
001132              LIntToHex(ORD(paragraph), @str);
001133              WriteLn('$$$ In ImageWith: paragraph,lp = (',str, ',', lp:1, ')' );
001134              END;
001135          {$ENDC}
001136
001137          altParaImage := NIL;
001138          s := paragraph.images.Scanner;
001139          WHILE s.Scan(paraImage) DO
001140              IF ((paraImage.textImage.headTxtImg = SELF.headTxtImg) AND
001141                  (paraImage.startLP <= lp) AND
001142                  (lp <= paraImage.endLP)) THEN
001143                      IF lp = paraImage.endLP THEN
001144                          IF paraImage.textImage <> SELF THEN
001145                              altParaImage := paraImage
001146                          ELSE
001147                              s.Done
```

```
001148                      ELSE
001149                          s.Done;
001150
001151          IF paraImage = NIL THEN
001152              paraImage := altParaImage;
001153          (*
001154          i := 1;
001155          WITH paragraph DO
001156              {$R-}
001157              WHILE (i <= numImages) DO
001158                  IF ((images[i].textImage.headTxtImg = SELF.headTxtImg) AND
001159                      (images[i].startLP <= lp) AND
001160                      (lp <= images[i].endLP)) THEN
001161                          BEGIN
001162                          paraImage := images[i];
001163                          i := MAXINT;
001164                          END
001165                  ELSE
001166                      i := i + 1;
001167              {$IFC fRngText}{$R+}{$ENDC}
001168          *)
001169
001170          {$IFC fTextTrace}
001171          IF fTextTrace THEN
001172              BEGIN
001173              LIntToHex(ORD(paraImage), @str);
001174              WriteLn('$$$ In ImageWith: paraImage found= ',str);
001175              END;
001176          {$ENDC}
001177
001178          ImageWith := paraImage;
001179          {$IFC fTrace}EP;{$ENDC}
001180      END;
001181
001182
001183  {$S SgTxtHot}
001184      FUNCTION TTextImage.ImageBottom: LONGINT;
001185      BEGIN
001186          {$IFC fTrace}BP(9);{$ENDC}
001187          IF SELF.imageList.Size > 0 THEN
001188              ImageBottom := TParaImage(SELF.imageList.Last).extentLRect.bottom
001189          ELSE
001190              ImageBottom := SELF.extentLRect.top;
001191          {$IFC fTrace}EP;{$ENDC}
001192      END;
001193
001194
001195  {$S SgTxtCld}
```

```
001196      PROCEDURE TTextImage.InsertNewPara(existingPara, newPara: TEditPara);
001197      VAR paraImage:      TParaImage;
001198          lastImage:      TParaImage;
001199          newParaImage:   TParaImage;
001200          textImage:      TTextImage;
001201          {$IFC fTextTrace}
001202          str:            STR255;
001203          {$ENDC}
001204      BEGIN
001205          {$IFC fTrace}BP(10);{$ENDC}
001206          {Try to find the image with the proper lp, but, failing that, see if there is any image that
001207           points to existingPara}
001208          paraImage := SELF.ImageWith(existingPara, existingPara.size - 1);
001209          IF paraImage = NIL THEN
001210              paraImage := SELF.ImageWith(existingPara, 0);
001211          IF paraImage = NIL THEN
001212              BEGIN
001213              {$IFC fTextTrace}
001214              IF fTextTrace THEN
001215                  BEGIN
001216                  LIntToHex(ORD(existingPara), @str);
001217                  WriteLn('@@@ In InsertNewPara: existingPara = (',str,')' );
001218                  WriteLn('@@@   ImageWith returned NIL!!');
001219                  END;
001220              {$ENDC}
001221              END
001222          ELSE
001223              BEGIN
001224              textImage := paraImage.textImage;
001225              lastImage := TParaImage(textImage.imageList.Last);
001226              newParaImage := textImage.NewParaImage(newPara, paraImage.extentLRect, 0, 0);
001227              textImage.imageList.InsAt(textImage.imageList.Pos(0, paraImage) + 1, newParaImage);
001228
001229              {If we inserted the new paraImage at the end of the current image list and if the
001230               last paragraph was previously split between two or more paraImages, then set the paragraph
001231               field in the first image of the next text image to the new paragraph, and adjust the
001232               paragraphs' images accordingly}
001233              IF (paraImage = lastImage) THEN
001234                  BEGIN
001235                  textImage := paraImage.textImage.nextTxtImg;
001236                  WHILE (textImage <> NIL) DO
001237                      IF textImage.imageList.Size > 0 THEN
001238                          BEGIN
001239                          paraImage := TParaImage(textImage.imageList.First);
001240                          IF paraImage.paragraph = lastImage.paragraph THEN
001241                              BEGIN
001242                              paraImage.paragraph := newPara;
001243                              newPara.InsImage(paraImage);     {|}
```

```
001244                          lastImage.paragraph.DelImage(paraImage, FALSE); {|}
001245                          textImage := textImage.nextTxtImg;
001246                          END
001247                      ELSE
001248                          textImage := NIL
001249                      END
001250                  ELSE
001251                      textImage := NIL
001252              END;
001253          END;
001254      {$IFC fTrace}EP;{$ENDC}
001255  END;
001256
001257
001258  {$S SgTxtCld}
001259      PROCEDURE TTextImage.InvalAll;
001260      VAR r:          LRect;
001261          textImage:  TTextImage;
001262
001263          PROCEDURE Inval(obj: TObject);
001264          BEGIN
001265              TParaImage(obj).InvalLinesWith(0, MAXINT);
001266          END;
001267      BEGIN
001268          {$IFC fTrace}BP(10);{$ENDC}
001269          textImage := SELF.headTxtImg;
001270          WHILE textImage <> NIL DO
001271              BEGIN
001272              textImage.imageList.Each(Inval);
001273              r := textImage.extentLRect;
001274              InsetLRect(r, -1, 0);
001275              SELF.view.panel.InvalLRect(r);
001276              textImage := textImage.nextTxtImg;
001277              END;
001278          {$IFC fTrace}EP;{$ENDC}
001279      END;
001280
001281
001282  {$S SgTxtCld}
001283      PROCEDURE TTextImage.Invalidate;
001284      VAR r:          LRect;
001285          s, sLine:   TListScanner;
001286          paraImage:  TParaImage;
001287          lineInfo:   TLineInfo;
001288      BEGIN
001289          {$IFC fTrace}BP(10);{$ENDC}
001290          s := SELF.imageList.Scanner;
001291          WHILE s.Scan(paraImage) DO
```

```
001292              IF paraImage.wasOffset THEN
001293                  BEGIN
001294                  r := paraImage.extentLRect;
001295                  InsetLRect(r, -1, 0);
001296                  thePad.InvalLRect(r);
001297                  paraImage.wasOffset := FALSE;
001298                  paraImage.changed := FALSE;
001299                  END
001300              ELSE IF paraImage.changed THEN
001301                  BEGIN
001302                  paraImage.changed := FALSE;
001303                  sLine := paraImage.lineList.Scanner;
001304                  WHILE sLine.Scan(lineInfo) DO
001305                      IF NOT lineInfo.valid THEN
001306                          BEGIN
001307                          lineInfo.valid := TRUE;
001308                          r := lineInfo.lineLRect;
001309                          r.left := paraImage.extentLRect.left;
001310                          r.right := paraImage.extentLRect.right;
001311                          InsetLRect(r, -1, 0);
001312                          thePad.InvalLRect(r);
001313                          END;
001314                  END;
001315
001316          IF NOT EmptyLRect(SELF.updateLRect) THEN
001317              BEGIN
001318              thePad.InvalLRect(SELF.updateLRect);
001319              SELF.updateLRect := zeroLRect;
001320              END;
001321
001322          IF SELF.nextTxtImg <> NIL THEN
001323              SELF.nextTxtImg.Invalidate;
001324          {$IFC fTrace}EP;{$ENDC}
001325      END;
001326
001327
001328  {$S SgTxtHot}
001329      PROCEDURE TTextImage.MarkChanged(startIndex: LONGINT; startLP: INTEGER;
001330                                                  endIndex: LONGINT; endLP: INTEGER);
001331      VAR sImg:       TListScanner;
001332          firstImage: TParaImage;
001333          lastImage:  TParaImage;
001334          paraImage:  TParaImage;
001335          found:      BOOLEAN;
001336          finished:   BOOLEAN;
001337          textImage:  TTextImage;
001338          tempLP:     INTEGER;
001339          stillOkay:  BOOLEAN;
```

```
001340      BEGIN
001341          {$IFC fTrace}BP(10);{$ENDC}
001342          stillOkay := TRUE;
001343          IF (startIndex = endIndex) AND (startLP = endLP) THEN
001344              BEGIN
001345              firstImage := SELF.ImageWith(TEditPara(SELF.text.paragraphs.At(startIndex)), startLP);
001346              lastImage := firstImage;
001347              stillOkay := firstImage <> NIL;
001348              END
001349          ELSE
001350              BEGIN
001351              SELF.GetImageRange(startIndex, startLP, endIndex, endLP, firstImage, lastImage);
001352              IF firstImage = NIL THEN
001353                  IF lastImage = NIL THEN
001354                      stillOkay := FALSE
001355                  ELSE
001356                      BEGIN
001357                      firstImage := lastImage;
001358                      startLP := endLP;
001359                      END
001360              ELSE IF lastImage = NIL THEN
001361                      BEGIN
001362                      lastImage := firstImage;
001363                      endLP := startLP;
001364                      END;
001365              END;
001366
001367          IF stillOkay THEN
001368              BEGIN
001369              IF firstImage = lastImage THEN
001370                  firstImage.InvalLinesWith(startLP, endLP)
001371              ELSE
001372                  BEGIN
001373                  textImage := firstImage.textImage;
001374                  found := FALSE;
001375                  finished := FALSE;
001376                  WHILE NOT finished AND (textImage <> NIL) DO
001377                      BEGIN
001378                      sImg := textImage.imageList.Scanner;
001379                      WHILE sImg.Scan(paraImage) DO
001380                          BEGIN
001381                          found := found OR (paraImage = firstImage);
001382                          IF found THEN
001383                              BEGIN
001384                              IF paraImage = lastImage THEN
001385                                  BEGIN
001386                                  tempLP := endLP;
001387                                  finished := TRUE;
```

```
001388                              sImg.Done;
001389                              END
001390                          ELSE
001391                              tempLP := paraImage.endLP;
001392                          paraImage.InvalLinesWith(startLP, tempLP);
001393                          startLP := 0;
001394                          END;
001395                      END;
001396                  textImage := textImage.nextTxtImg;
001397                  END;
001398              END;
001399          END;
001400      {$IFC fTrace}EP;{$ENDC}
001401  END;
001402
001403
001404  {$S SgTxtHot}
001405      PROCEDURE TTextImage.MousePress(mouseLPt: LPoint);
001406      VAR currParaImage:  TParaImage;
001407          currLP:         INTEGER;
001408          textImage:      TTextImage;
001409          firstTxtImg:    TTextImage;
001410          selection:      TSelection;
001411          paraIndex:      LONGINT;
001412
001413      BEGIN
001414          {$IFC fTrace}BP(10);{$ENDC}
001415          selection := SELF.view.panel.selection;
001416          WHILE selection.coSelection <> NIL DO
001417              selection := selection.coSelection;
001418          IF (clickState.fShift OR (clickState.clickCount > 1)) AND InClass(selection, TTextSelection) THEN
001419              { let the selection extend itself }
001420              selection.MousePress(mouseLPt)
001421          ELSE
001422              BEGIN
001423              textImage := SELF.FindTextImage(mouseLPt, firstTxtImg);
001424              textImage.FindParaAndLp(mouseLPt, currParaImage, paraIndex, currLP);
001425
001426              {If we are a coSelection then BeginSelection should already have been called when
001427               panel selection was created}
001428              IF SELF.view.panel.selection.coSelection = NIL THEN
001429                  SELF.view.panel.BeginSelection
001430              ELSE
001431                  selection.Highlight(hOnToOff);
001432
001433              selection := selection.FreedAndReplacedBy(TInsertionPoint.CREATE(NIL,
001434                                                  SELF.Heap, SELF.view, textImage, mouseLPt,
001435                                                  currParaImage.paragraph, paraIndex, currLP));
```

```
001436              SELF.text.ChangeSelInOtherPanels(TTextSelection(selection));
001437              SELF.text.HiliteParagraphs(hOffToOn, paraIndex, currLP, paraIndex, currLP, FALSE);
001438              END;
001439          {$IFC fTrace}EP;{$ENDC}
001440      END;
001441
001442
001443  {$S SgTxtHot}
001444      FUNCTION TTextImage.NewEditPara(initialSize: INTEGER; itsFormat: TParaFormat): TEditPara;
001445      BEGIN
001446          {$IFC fTrace}BP(10);{$ENDC}
001447          NewEditPara := TEditPara.CREATE(NIL, SELF.Heap, initialSize, itsFormat);
001448          {$IFC fTrace}EP;{$ENDC}
001449      END;
001450
001451
001452  {$S SgTxtCld}
001453      FUNCTION TTextImage.NewParaImage(itsParagraph: TEditPara; itsLRect: LRect;
001454                                       lineTop: LONGINT; lineLeft: LONGINT): TParaImage;
001455      VAR paraImage:  TParaImage;
001456      BEGIN
001457          {$IFC fTrace}BP(10);{$ENDC}
001458          paraImage := TParaImage.CREATE(NIL, SELF.Heap, SELF.view, itsParagraph, itsLRect, lineTop, lineLeft);
001459          paraImage.textImage := SELF;
001460          itsParagraph.InsImage(paraImage);
001461          NewParaImage := paraImage;
001462          {$IFC fTrace}EP;{$ENDC}
001463      END;
001464
001465
001466  {$S SgTxtCld}
001467      FUNCTION TTextImage.NewTextImage(heap: THeap; itsView: TView; itsLRect: LRect;
001468                                           itsText:TText; isGrowable: BOOLEAN): TTextImage;
001469      BEGIN
001470          {$IFC fTrace}BP(10);{$ENDC}
001471          NewTextImage := TTextImage.CREATE(NIL, heap, itsView, itsLRect, itsText, isGrowable);
001472          {$IFC fTrace}EP;{$ENDC}
001473      END;
001474
001475
001476  {$S SgTxtHot}
001477      FUNCTION TTextImage.NewTextSelection(firstPara: TEditPara; firstIndex: LONGINT; firstLP: INTEGER;
001478                                           lastPara: TEditPara; lastIndex: LONGINT; lastLP: INTEGER
001479                                           ): TTextSelection;
001480      VAR textSel:    TTextSelection;
001481          heap:       THeap;
001482          view:       TView;
001483      BEGIN
```

```
001484          {$IFC fTrace}BP(10);{$ENDC}
001485          heap := SELF.Heap;
001486          view := SELF.view;
001487          IF firstPara = lastPara THEN
001488              IF firstLP = lastLP THEN
001489                  textSel := TInsertionPoint.CREATE(NIL, heap, view, SELF, zeroLPt,
001490                                                    firstPara, firstIndex, firstLP)
001491              ELSE
001492                  textSel := TOneParaSelection.CREATE(NIL, heap, view, SELF, zeroLPt,
001493                                                    firstPara, firstIndex, firstLP, lastLP)
001494          ELSE
001495              textSel := TMultiParaSelection.CREATE(NIL, heap, view, SELF, zeroLPt,
001496                                                    firstPara, firstIndex, firstLP,
001497                                                    lastPara, lastIndex, lastLP, TRUE);
001498          NewTextSelection := textSel;
001499          {$IFC fTrace}EP;{$ENDC}
001500      END;
001501
001502
001503  {$S SgTxtCld}
001504      PROCEDURE TTextImage.OffsetBy(deltaLPt: LPoint);
001505      { Can be used to quickly move a text image }
001506          PROCEDURE OffsetImage(obj: TObject);
001507          BEGIN
001508              TParaImage(obj).OffsetBy(deltaLPt);
001509          END;
001510      BEGIN
001511          {$IFC fTrace}BP(10);{$ENDC}
001512          WITH deltaLPt DO
001513              {$H-} OffsetLRect(SELF.extentLRect, h, v); {$H+}
001514          SELF.imageList.Each(OffsetImage);
001515          {$IFC fTrace}EP;{$ENDC}
001516      END;
001517
001518
001519  {$S SgTxtHot}
001520      PROCEDURE TTextImage.RecomputeImages(drawAction: TDrawAction; invalBits: BOOLEAN);
001521      LABEL   1;
001522      VAR drawLRect:  LRect;
001523          lastLP:         INTEGER;
001524          lastDrawnImage: TParaImage;
001525          nextTxtImg:     TTextImage;
001526          paraImage:      TParaImage;
001527          s:              TListScanner;
001528          tempList:       TList;
001529          beginAtLP:      INTEGER;
001530          returnLRect:    LRect;
001531          lastImage:      TParaImage;
```

```
001532          firstImage:     TParaImage;
001533          lastOneChanged: BOOLEAN;
001534          deltaLPt:       LPoint;
001535          currParaIndex:  LONGINT;
001536          paragraph:      TEditPara;
001537          newBottom:      LONGINT;
001538          realAction:     TDrawAction;
001539          r:              LRect;
001540          {$IFC fTextTrace}
001541          str:            STR255;
001542          {$ENDC}
001543
001544          FUNCTION OnlyPartDrawn(pImage: TParaImage): BOOLEAN;
001545          VAR wontFit:    BOOLEAN;
001546              sLine:      TListScanner;
001547              deleteRest: BOOLEAN;
001548              lineInfo:   TLineInfo;
001549
001550              PROCEDURE DrawPImage(obj: TObject);
001551              VAR action: TDrawAction;
001552                  bits:   BOOLEAN;
001553              BEGIN
001554                  {$IFC fTrace}BP(10);{$ENDC}
001555                  bits := invalBits;
001556                  action := realAction;
001557                  {If the paragraph was offset, we don't want DrawPara to draw the changed lines, so
001558                   we display the offset paragraph case below and pass actionNone to DrawPara.
001559                   However, we must pass FALSE for invalBits since we still need the wasOffset flag
001560                   set for the display code below}
001561                  IF pImage.wasOffset THEN
001562                      BEGIN
001563                      action := actionNone;
001564                      bits := FALSE;
001565                      END;
001566                  pImage.DrawParaImage(drawLRect, beginAtLP, action, bits, returnLRect);
001567                  {$IFC fTrace}EP;{$ENDC}
001568              END;
001569          BEGIN
001570              {$IFC fTrace}BP(10);{$ENDC}
001571              wontFit := FALSE;
001572              lastOneChanged := FALSE;
001573              pImage.textImage := SELF;
001574              {$IFC fTextTrace}
001575              IF fTextTrace THEN
001576                  BEGIN
001577                  LIntToHex(ORD(pImage), @str);
001578                  WriteLn('++ Entering OnlyPartDrawn: pImage =', str);
001579                  WriteLn('++                       : deltaLPt.v =', deltaLPt.v:1, '  drawLRect.top = ',
```

```
001580                                          drawLRect.top:1);
001581              END;
001582          {$ENDC}
001583
001584          {offset pImage before recalculating so that unchanged lineInfo's get offset}
001585          IF drawLRect.top <> pImage.extentLRect.top THEN
001586              BEGIN
001587              deltaLPt.v := drawLRect.top - pImage.extentLRect.top;
001588              pImage.OffsetBy(deltaLPt);
001589              pImage.wasOffset := TRUE;   {so that we know what to redraw/invalidate}
001590              {If we moved the last paraImage up, maybe more will fit, so force call to DrawPara
001591               by setting changed TRUE}
001592              IF (deltaLPt.v < 0) AND (pImage = SELF.imageList.Last) THEN
001593                  BEGIN
001594                  lastOneChanged := TRUE;
001595                  pImage.changed := TRUE;
001596                  END;
001597              END;
001598
001599          IF pImage.changed THEN
001600              BEGIN
001601              lastLP := pImage.endLP;
001602              {$IFC fTextTrace}
001603              IF fTextTrace THEN
001604                  WriteLn('++ OnlyPartDrawn: pImage changed, about to call DrawPara; old endLP =',
001605                                              lastLP:1);
001606              {$ENDC}
001607
001608              SELF.FilterAndDo(pImage, DrawPImage);
001609
001610              {$IFC fTextTrace}
001611              IF fTextTrace THEN
001612                  WriteLn('++ OnlyPartDrawn: DrawPara just called; pImage.endLP =',
001613                                              pImage.endLP:1, ' para size = ', pImage.paragraph.size:1);
001614              {$ENDC}
001615              lastOneChanged := lastOneChanged OR (pImage.endLP <> lastLP);
001616              END
001617          ELSE IF deltaLPt.v > 0 THEN
001618              BEGIN
001619              {$IFC fTextTrace}
001620              IF fTextTrace THEN
001621                  WriteLn('++ OnlyPartDrawn: pImage.extentLRect.bottom =', pImage.extentLRect.bottom:1,
001622                              '  drawLRect.bottom = ', drawLRect.bottom:1);
001623              {$ENDC}
001624
001625              wontFit := pImage.extentLRect.bottom > drawLRect.bottom;
001626              IF wontFit THEN
001627                  BEGIN
```

```
001628                  {Ideally, if textImages are same width, just insert extra lineInfo's
001629                   into first paraImage of next textImage; If they are not same width, we
001630                   still insert them but mark them invalid. For now just delete them}
001631                  lastOneChanged := TRUE;
001632                  sLine := pImage.lineList.Scanner;
001633                  deleteRest := FALSE;
001634                  WHILE sLine.Scan(lineInfo) DO
001635                      IF deleteRest THEN
001636                          sLine.Delete(TRUE)
001637                      ELSE IF lineInfo.lineLRect.bottom > drawLRect.bottom THEN
001638                          BEGIN
001639                          {If the first lineInfo won't fit then set pImage.endLP to -1, indicating that
001640                           none of the paragraph fit}
001641                          IF sLine.Position = 1 THEN
001642                              BEGIN
001643                              sLine.Done;
001644                              pImage.endLP := -1;
001645                              END
001646                          ELSE
001647                              BEGIN
001648                              pImage.extentLRect.bottom := lineInfo.lineLRect.top;
001649                              pImage.endLP := lineInfo.startLP;
001650                              sLine.Delete(TRUE);
001651                              deleteRest := TRUE;
001652                              END;
001653                          END;
001654                  END;
001655              END;
001656
001657          drawLRect.top := pImage.extentLRect.bottom;
001658          lastLP := pImage.endLP;
001659          wontFit := wontFit OR (pImage.endLP <> pImage.paragraph.size);
001660          IF pImage.wasOffset AND (lastLP >= 0) THEN
001661              BEGIN
001662              r := pImage.extentLRect;
001663              InsetLRect(r, -1, 0);
001664              IF realAction = actionDraw THEN
001665                  BEGIN
001666                  FillLRect(r, lPatWhite);
001667                  pImage.RedrawLines(0, MAXINT);
001668                  END
001669              ELSE IF realAction = actionInval THEN
001670                  thePad.InvalLRect(r);
001671              pImage.wasOffset := NOT invalBits;
001672              pImage.changed := FALSE;
001673              END;
001674
001675          {after the first paragraph is drawn, reset beginAtLP}
```

```
001676              beginAtLP := 0;
001677              IF NOT wontFit THEN
001678                  currParaIndex := currParaIndex + 1;
001679              {$IFC fTextTrace}
001680              IF fTextTrace THEN
001681                  WriteLn('++ EXITING OnlyPartDrawn: deltaLPt.v =', deltaLPt.v:1, '  wontFit = ', wontFit);
001682              {$ENDC}
001683
001684
001685              {return TRUE when a paragraph does not get completely displayed}
001686              OnlyPartDrawn := wontFit;
001687              {$IFC fTrace}EP;{$ENDC}
001688          END;
001689
001690      BEGIN {RecomputeIMages}
001691          {$IFC fTrace}BP(10);{$ENDC}
001692          IF SELF = SELF.headTxtImg THEN
001693              SELF.SetFirstIndex;
001694          currParaIndex := SELF.firstIndex;
001695          drawLRect := SELF.extentLRect;
001696          IF SELF.growsDynamically THEN
001697              drawLRect.bottom := SELF.view.extentLRect.bottom;
001698          realAction := drawAction;
001699          IF drawAction = actionDraw THEN
001700              IF NOT SELF.view.OKToDrawIn(drawLRect) OR deferUpdate THEN
001701                  realAction := actionInval;
001702
001703          beginAtLP := SELF.startLP;
001704          deltaLPt := zeroLPt;
001705
001706          {Recompute paragraphs until we reach the end of our imageList or no more will fit}
001707          {Use GOTO so we can hold on to the scanner if needed}
001708          s := SELF.imageList.Scanner;
001709          WHILE s.Scan(lastDrawnImage) DO
001710              IF OnlyPartDrawn(lastDrawnImage) THEN
001711                  GOTO 1;
001712
001713          1:
001714          {$IFC fTextTrace}
001715          IF fTextTrace THEN
001716              BEGIN
001717              LIntToHex(ORD(lastDrawnImage), @str);
001718              WriteLn('++ RecomputeImages: Just fell out of OnlyPartDrawn loop: lastDrawnImage=',
001719                              str, '  lastOneChanged=', lastOneChanged);
001720              END;
001721          {$ENDC}
001722
001723          {At this point, lastDrawnImage will be NIL iff the scan went through the entire list
```

```
001724          and OnlyPartDrawn never returned TRUE}
001725
001726          {Hence, if lastDrawnImage is NIL, then we displayed all our paraImages and there may
001727           still be some space left, so steal the next textImage's paraImages (if any).  Continue
001728           this in the Repeat loop until we use up the space or exhaust the paraImages.  Note that we
001729           don't need to check nextTxtImg if none of the paraImages changed and had to be recalculated}
001730          nextTxtImg := SELF.nextTxtImg;
001731          IF lastDrawnImage = NIL THEN
001732              BEGIN
001733              IF lastOneChanged OR TRUE THEN  {<-- temporary!!??}
001734                  REPEAT
001735                      IF nextTxtImg <> NIL THEN
001736                          BEGIN
001737                          s := nextTxtImg.imageList.Scanner;
001738
001739                          {Delete the first paraImage in the next textImage if it pointed to the same
001740                           paragraph as the current textImage's last paraImage, since, if we got to
001741                           this point, we must have already displayed the whole paragraph}
001742                          IF (nextTxtImg.imageList.Size > 0) AND (SELF.imageList.Size > 0) THEN
001743                              BEGIN
001744                              firstImage := TParaImage(nextTxtImg.imageList.First);
001745                              lastImage := TParaImage(SELF.imageList.Last);
001746                              IF firstImage.paragraph = lastImage.paragraph THEN
001747                                  IF s.Scan(paraImage) THEN
001748                                      BEGIN
001749                                      paraImage.paragraph.images.DelObject(paraImage, FALSE);
001750                                      s.Delete(TRUE);
001751                                      END;
001752                              END;
001753                          WHILE s.Scan(lastDrawnImage) DO
001754                              BEGIN
001755                              lastDrawnImage.InvalLinesWith(0, MAXINT);
001756                              deltaLPt.v := 0;
001757                              IF OnlyPartDrawn(lastDrawnImage) THEN
001758                                  BEGIN
001759                                  {if we didn't stop between paragraphs, install the paraImage in our list,
001760                                   replace the next textImage's first paraImage with a copy of this one,
001761                                   then terminate the scan.  Note that the extentLRect and
001762                                   height fields of the next TextImage's first paraImage are now incorrect,
001763                                   but will be rectified when nextTxtImg.RecomputeImage is called}
001764                                  IF (lastLP >= 0) THEN
001765                                      BEGIN
001766                                      SELF.imageList.InsLast(lastDrawnImage);
001767                                      paraImage := nextTxtImg.NewParaImage(lastDrawnImage.paragraph,
001768                                                                  lastDrawnImage.extentLRect, 0, 0);
001769                                      s.Replace(paraImage, FALSE);
001770                                      END;
001771                                  s.Done;
```

```
001772                              END
001773                          ELSE
001774                              BEGIN
001775                              {remove lastDrawnImage from nextTxtImg.imageList and install in
001776                               SELF.imageList}
001777                              s.Delete(FALSE);
001778                              SELF.imageList.InsLast(lastDrawnImage);
001779                              END;
001780                          END;
001781                      IF lastDrawnImage = NIL THEN
001782                          nextTxtImg := nextTxtImg.nextTxtImg;
001783                      END;
001784                UNTIL (lastDrawnImage <> NIL) OR (nextTxtImg = NIL);
001785
001786            IF lastDrawnImage = NIL THEN
001787                BEGIN
001788                {We exhausted all of the images and there is still potentially some room, so
001789                 look at the paragraph list and see if there are some paragraphs for which no
001790                 paraImages have yet been generated.
001791                 NOTE: this is where initial imaging of text without paraImages will be routed}
001792                IF currParaIndex <= SELF.text.paragraphs.Size THEN
001793                    BEGIN
001794                    s := SELF.text.paragraphs.ScannerFrom(currParaIndex-1, scanForward);
001795                    WHILE s.Scan(paragraph) DO
001796                        BEGIN
001797                        lastDrawnImage := SELF.NewParaImage(paragraph, drawLRect, 0, 0);
001798                        IF OnlyPartDrawn(lastDrawnImage) THEN
001799                            s.Done;
001800                        IF lastdrawnImage.endLP >= 0 THEN
001801                            SELF.imageList.InsLast(lastDrawnImage)
001802                        ELSE
001803                            BEGIN
001804                            lastDrawnImage.paragraph.images.DelObject(lastDrawnImage, TRUE);
001805                            lastDrawnImage := NIL;
001806                            END;
001807                        END;
001808                    END;
001809                END;
001810            END
001811        ELSE IF nextTxtImg <> NIL THEN
001812            BEGIN
001813            IF lastOneChanged THEN
001814                BEGIN
001815                {we stopped displaying in the middle of a paragraph, so give the rest of our
001816                 paraImages to the next textImage (note that the scanner (s) is still valid
001817                 because we jumped out of scan loop above)}
001818                {put rest of SELF.imageList paraImages in tempList, then insert tempList into nextTxtImg}
001819                tempList := TList.CREATE(NIL, SELF.Heap, 0);
```

```
001820                  {if we didn't display any of the current paraImage then delete it from this list}
001821                  IF lastLP < 0 THEN
001822                      BEGIN
001823                      {$IFC fTextTrace}
001824                      IF fTextTrace THEN
001825                          BEGIN
001826                          LIntToHex(ORD(lastDrawnImage), @str);
001827                          WriteLn('++ RecomputeImages: lastLP < 0; lastdrawnImage=', str);
001828                          END;
001829                      {$ENDC}
001830                      s.Delete(FALSE);
001831                      paraImage := lastDrawnImage;
001832                      paraImage.endLP := 0;
001833                      END
001834                  ELSE
001835                      BEGIN
001836                      paraImage := nextTxtImg.NewParaImage(lastDrawnImage.paragraph,
001837                                                           lastDrawnImage.extentLRect, 0, 0);
001838                      {$IFC fTextTrace}
001839                      IF fTextTrace THEN
001840                          BEGIN
001841                          LIntToHex(ORD(paraImage), @str);
001842                          WriteLn('++ RecomputeImages: copy of lastDrawnImage =', str);
001843                          END;
001844                      {$ENDC}
001845                      END;
001846                  tempList.InsLast(paraImage);
001847                  {put the paraImages into tempList in reverse order so that we can scan it and insert
001848                   the images at the beginning of nextTxtImg.imageList (a double-reverse)}
001849                  WHILE s.Scan(paraImage) DO
001850                      BEGIN
001851                      {$IFC fTextTrace}
001852                      IF fTextTrace THEN
001853                         BEGIN
001854                         LIntToHex(ORD(paraImage), @str);
001855                         WriteLn('++ RecomputeImages: appending to tempList and deleting from SELF pImg=', str);
001856                         END;
001857                      {$ENDC}
001858                      tempList.InsFirst(paraImage);
001859                      s.Delete(FALSE);
001860                      END;
001861
001862                  {Delete the last paraImage inserted in the tempList if it pointed to the same
001863                   paragraph as the next textImage's first paraImage}
001864                  IF nextTxtImg.imageList.Size > 0 THEN
001865                      BEGIN
001866                      firstImage := TParaImage(nextTxtImg.imageList.First);
001867                      lastImage := TParaImage(tempList.First);
```

```
001868                  IF firstImage.paragraph = lastImage.paragraph THEN
001869                      BEGIN
001870                      firstImage.InvalLinesWith(0, MAXINT);
001871                     {$IFC fTextTrace}
001872                      IF fTextTrace THEN
001873                          BEGIN
001874                          LIntToHex(ORD(lastImage), @str);
001875                          WriteLn('++ RecomputeImages: ', str,
001876                              ' points to same para as nxtTxt.firstImg, so tempList.del');
001877                          END;
001878                      {$ENDC}
001879                      IF lastDrawnImage = lastImage THEN
001880                          lastDrawnImage := NIL;
001881                      lastImage.paragraph.images.DelObject(lastImage, FALSE);
001882                      tempList.DelFirst(TRUE);
001883                      END;
001884                  END;
001885              {$IFC fTextTrace}
001886              IF fTextTrace THEN
001887                  WriteLn('++ RecomputeImages: About to insert rest of pimages into nextTImage; list size=',
001888                              tempList.Size:1);
001889              {$ENDC}
001890              s := tempList.Scanner;
001891              WHILE s.Scan(paraImage) DO
001892                  BEGIN
001893                  paraImage.textImage := nextTxtImg;
001894                  paraImage.InvalLinesWith(0, MAXINT);
001895                  nextTxtImg.imageList.InsFirst(paraImage);
001896                  s.Delete(FALSE);
001897                  END;
001898              tempList.Free;
001899              END; {lastOneChanged}
001900          END {nxtTxtImg <> NIL}
001901      ELSE
001902          {If we have stopped displaying in the middle of a paragraph and there is no next
001903           text image to display the excess, then delete the remaining paraImages}
001904          BEGIN
001905          IF lastLP < 0 THEN
001906              BEGIN
001907              lastDrawnImage.paragraph.images.DelObject(lastDrawnImage, FALSE);
001908              s.Delete(TRUE);
001909              END;
001910          WHILE s.Scan(paraImage) DO
001911              BEGIN
001912              paraImage.paragraph.images.DelObject(paraImage, FALSE);
001913              s.Delete(TRUE);
001914              END;
001915          END;
```

```
001916
001917          {Set up new bottom and erase any garbage due to text moving up}
001918          newBottom := SELF.ImageBottom;
001919          IF newBottom < SELF.formerBottom THEN
001920              WITH SELF, extentLRect DO
001921                  {$H-}
001922                  BEGIN
001923                  SetLRect(updateLRect, left-1, newBottom, right+1, formerBottom);
001924                  IF realAction = actionDraw THEN
001925                      FillLRect(updateLRect, lPatWhite)
001926                  ELSE IF realAction = actionInval THEN
001927                      thePad.invalLRect(updateLRect);
001928                  IF invalBits THEN
001929                      updateLRect := zeroLRect;
001930                  END;
001931                  {$H+}
001932          SELF.formerBottom := newBottom;
001933          IF SELF.growsDynamically THEN
001934              WITH SELF, extentLRect DO
001935                  {$H-}
001936                  bottom := Max(newBottom, top + minHeight);
001937                  {$H+}
001938
001939
001940          {Now tell the next textImage in the chain to recompute itself}
001941          nextTxtImg := SELF.nextTxtImg;
001942          IF nextTxtImg <> NIL THEN
001943              BEGIN
001944              {$IFC fTextTrace}
001945              IF fTextTrace THEN
001946                  WriteLn('++ RecomputeImages: About to call RecomputeImages for nextTxtImg; nTI.imgLst.Size=',
001947                                  nextTxtImg.imageList.Size:1);
001948              {$ENDC}
001949              IF lastDrawnImage = NIL THEN
001950                  nextTxtImg.startLP := 0
001951              ELSE
001952                  {$H-} nextTxtImg.startLP := Max(0, lastDrawnImage.endLP); {$H+}
001953              nextTxtImg.firstIndex := currParaIndex;
001954              nextTxtImg.RecomputeImages(drawAction, invalBits);
001955              END;
001956
001957          {$IFC fTrace}EP;{$ENDC}
001958      END; {RecomputeImages}
001959
001960
001961  {$S SgTxtCld}
001962      PROCEDURE TTextImage.Resize(newExtent: LRect);
001963      BEGIN
```

```
001964          {$IFC fTrace}BP(10);{$ENDC}
001965          SELF.extentLRect := newExtent;
001966          SELF.InvalAll;
001967          {$IFC fTrace}EP;{$ENDC}
001968      END;
001969
001970
001971  {$S SgTxtHot}
001972      FUNCTION TTextImage.SeesSameAs(image: TImage): BOOLEAN;
001973      BEGIN
001974          {$IFC fTrace}BP(9);{$ENDC}
001975          IF SELF = image THEN
001976              SeesSameAs := TRUE
001977          ELSE IF InClass(image, TParaImage) THEN
001978              SeesSameAs := SELF.text = TParaImage(image).textImage.text
001979          ELSE IF InClass(image, TTextImage) THEN
001980              SeesSameAs := SELF.text = TTextImage(image).text
001981          ELSE
001982              SeesSameAs := FALSE;
001983          {$IFC fTrace}EP;{$ENDC}
001984      END;
001985
001986  {$S SgTxtHot}
001987      PROCEDURE TTextImage.SetFirstIndex;
001988      BEGIN
001989          {$IFC fTrace}BP(10);{$ENDC}
001990          SELF.firstIndex := 1;
001991          {$IFC fTrace}EP;{$ENDC}
001992      END;
001993
001994
001995  {$S SgTxtWrm}
001996      FUNCTION  TTextImage.TxtImgForClipBoard(heap: THeap; itsView: TView; itsLRect: LRect;
001997                                              itsText:TText; isGrowable: BOOLEAN): TTextImage;
001998      BEGIN
001999          {$IFC fTrace}BP(10);{$ENDC}
002000          TxtImgForClipBoard := TTextImage.CREATE(NIL, heap, itsView, itsLRect, itsText, isGrowable);
002001          {$IFC fTrace}EP;{$ENDC}
002002      END;
002003
002004  {$S SgTxtIni}
002005  END; {Methods of TTextImage}
002006
002007
002008  METHODS OF TTextView;
002009
002010  {$S SgTxtCld}
002011      FUNCTION TTextView.CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsExtent: LRect): TTextView;
```

```
002012      BEGIN
002013          {$IFC fTrace}BP(10);{$ENDC}
002014          IF object = NIL THEN
002015              object := NewObject(heap, THISCLASS);
002016          SELF := TTextView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, heap),
002017                                                  stdMargins, FALSE {, Damnit!}));
002018          SELF.textImage := NIL;
002019          SELF.valid := FALSE;
002020          {$IFC fTrace}EP;{$ENDC}
002021      END;
002022
002023  {$S SgTxtCld}
002024  {$IFC fTextTrace}
002025      PROCEDURE TTextView.Fields(PROCEDURE Field(nameAndType: S255));
002026      BEGIN
002027          SUPERSELF.Fields(Field);
002028          Field('textImage: TTextImage');
002029          Field('valid: BOOLEAN');
002030          Field('');
002031      END;
002032  {$ENDC}
002033
002034  {$S SgTxtCld}
002035  {$IFC fUseUnivText}
002036      PROCEDURE TTextView.CreateUniversalText;
002037      VAR univText:   TTextWriteUnivText;
002038      BEGIN
002039          {$IFC fTrace}BP(10);{$ENDC}
002040          IF NOT clipBoard.hasUniversalText THEN
002041              BEGIN
002042              univText := TTextWriteUnivText.CREATE(NIL, mainHeap, NIL, 512,
002043                                                      TTextSelection(SELF.panel.selection));
002044              univText.Free;
002045              END;
002046          {$IFC fTrace}EP;{$ENDC}
002047      END;
002048  {$ENDC}
002049
002050
002051  {$S SgTxtCld}
002052      FUNCTION TTextView.CursorAt(mouseLPt: LPoint): TCursorNumber;
002053      BEGIN
002054          {$IFC fTrace}BP(10);{$ENDC}
002055          IF LPtInLRect(mouseLPt, SELF.textImage.extentLRect) THEN
002056              CursorAt := textCursor
002057          ELSE
002058              CursorAt := arrowCursor;
002059          {$IFC fTrace}EP;{$ENDC}
```

```
002060      END;
002061
002062  {$S SgTxtCld}
002063      PROCEDURE TTextView.Draw;
002064      BEGIN
002065          {$IFC fTrace}BP(10);{$ENDC}
002066          IF NOT SELF.valid THEN
002067              BEGIN
002068              SELF.textImage.RecomputeImages(actionNone, TRUE);
002069              SELF.valid := TRUE;
002070              END;
002071          SELF.textImage.Draw;
002072          {$IFC fTrace}EP;{$ENDC}
002073      END;
002074
002075  {$S SgTxtCld}
002076      PROCEDURE TTextView.MousePress(mouseLPt: LPoint);
002077      BEGIN
002078          {$IFC fTrace}BP(10);{$ENDC}
002079          SELF.textImage.MousePress(mouseLPt);
002080          {$IFC fTrace}EP;{$ENDC}
002081      END;
002082
002083
002084  {$S SgTxtIni}
002085  END; {Methods of TTextView}
002086
002087  {$S SgTxtCld}
002088
002089  {$IFC fUseUnivText}
002090  METHODS OF TTextWriteUnivText;
002091
002092  {$S SgTxtCld}
002093      FUNCTION  TTextWriteUnivText.CREATE(object: TObject; heap: THeap;
002094                                          itsString: TString; itsDataSize: INTEGER;
002095                                          itsTextSel: TTextSelection): TTextWriteUnivText;
002096      BEGIN
002097          {$IFC fTrace}BP(10);{$ENDC}
002098          IF object = NIL THEN
002099              object := NewObject(heap, THISCLASS);
002100          WITH TTextWriteUnivText(object) DO
002101              BEGIN
002102              textSelection := itsTextSel;
002103              currIndex := 1;
002104              currPara := itsTextSel.textRange.firstPara;
002105              currLP := 0;
002106              currStyleIndex := 1;
002107              currTStyles := itsTextSel.textRange.firstPara.typeStyles;
```

```
002108              END;
002109
002110          SELF := TTKWriteUnivText.CREATE(object, heap, itsString, itsDataSize);
002111          {$IFC fTrace}EP;{$ENDC}
002112      END;
002113
002114  {$S SgTxtCld}
002115  {$IFC fTextTrace}
002116      PROCEDURE TTextWriteUnivText.Fields(PROCEDURE Field(nameAndType: S255));
002117      BEGIN
002118          SUPERSELF.Fields(Field);
002119          Field('textSelection: TTextSelection');
002120          Field('currIndex: LONGINT');
002121          Field('currPara: TEditPara');
002122          Field('currLP: INTEGER');
002123          Field('currStyleIndex: INTEGER');
002124          Field('currTStyles: TArray');
002125          Field('');
002126      END;
002127  {$ENDC}
002128
002129  {$S SgTxtCld}
002130      PROCEDURE TTextWriteUnivText.FillParagraph;
002131      VAR startPos:   INTEGER;
002132          endPos:     INTEGER;
002133          currChange: TStyleChange;
002134          nextChange: TStyleChange;
002135          numChars:   INTEGER;
002136      BEGIN
002137          {$IFC fTrace}BP(10);{$ENDC}
002138          {$IFC fTextTrace}
002139          IF fTextTrace THEN
002140              BEGIN
002141              WriteLn('<-> Entering FillRun: Current fields are:');
002142              WriteLn('<->        currIndex = ', SELF.currIndex:1, ' currStyleIndex = ', SELF.currStyleIndex:1);
002143              WriteLn('<->        currLP = ', SELF.currLP:1);
002144              END;
002145          {$ENDC}
002146          SELF.data.DelAll;
002147          IF SELF.currIndex <= SELF.textSelection.textRange.lastIndex THEN
002148              BEGIN
002149              WITH SELF.paragraphDescriptor, SELF.currPara.format DO
002150                  BEGIN
002151                  paragraphStart := SELF.textSelection.isParaSelection AND {LSR}
002152                                            (SELF.currLP = 0);
002153                  firstLineMargin := firstIndent;
002154                  bodyMargin := leftIndent;
002155                  rightMargin := rightMargin - rightIndent; {????}
```

```
002156                  paraLeading := spaceBelowPara;
002157                  END;
002158              IF SELF.currPara.size = 0 THEN
002159                  BEGIN
002160                  endPos := 0;
002161                  numChars := 0;
002162                  END
002163              ELSE
002164                  BEGIN
002165                  REPEAT
002166                      SELF.currTStyles.GetAt(SELF.currStyleIndex, @currChange);
002167                      WITH SELF.characterDescriptor DO
002168                          BEGIN
002169                          {$H-}
002170                          font := QDFontNumber(currChange.newStyle);
002171                          {$H+}
002172                          face := currChange.newStyle.onFaces;
002173                          END;
002174                      startPos := Max(SELF.currLP, currChange.lp) + 1;
002175                      SELF.currTStyles.GetAt(SELF.currStyleIndex+1, @nextChange);
002176                      endPos := Min(SELF.currPara.size, nextChange.lp);
002177                      numChars := endPos - startPos + 1;
002178                      IF numChars = 0 THEN
002179                          SELF.currStyleIndex := SELF.currStyleIndex + 1;
002180                  UNTIL numChars > 0;
002181                  SELF.data.InsManyAt(1, SELF.currPara, startPos, numChars);
002182                  END;
002183              IF endPos = SELF.currPara.size THEN
002184                  BEGIN
002185                  SELF.currIndex := SELF.currIndex + 1;
002186                  WITH SELF DO
002187                      IF currIndex <= textSelection.textRange.lastIndex THEN
002188                          BEGIN
002189                          currLP := 0;
002190                          currStyleIndex := 1;
002191                          {$H-}
002192                          currPara := TEditPara(SELF.textSelection.textImage.text.paragraphs.At(
002193                                                                  SELF.currIndex));
002194                          {$H+}
002195                          currTStyles := currPara.typeStyles;
002196                          {$H-}
002197                          SELF.data.InsAt(numChars+1, CHR(ascReturn)); {last statement in IF!!}
002198                          {$H+}
002199                          END
002200                      ELSE IF textSelection.isParaSelection THEN
002201                          {$H-}
002202                          SELF.data.InsAt(numChars+1, CHR(ascReturn)); {last statement in IF!!}
002203                          {$H+}
```

```
002204                  END
002205              ELSE
002206                  BEGIN
002207                  SELF.currLP := SELF.currLP + numChars;
002208                  SELF.paragraphDescriptor.additionalChrInParagraph :=  {LSR}
002209                          SELF.currPara.size - endPos + 1;
002210                  END;
002211              END;
002212          {$IFC fTextTrace}
002213          IF fTextTrace THEN
002214              BEGIN
002215              WriteLn;
002216              WriteLn('>-< EXITING FillRun: Current fields are:');
002217              WriteLn('>-<        currIndex = ', SELF.currIndex:1, ' currStyleIndex = ', SELF.currStyleIndex:1);
002218              WriteLn('>-<        currLP = ', SELF.currLP:1);
002219              WriteLn('-------------------------------------------------------------');
002220              END;
002221          {$ENDC}
002222          {$IFC fTrace}EP;{$ENDC}
002223      END;
002224
002225  {$S SgTxtIni}
002226  END; {Methods of TTextWriteUnivText}
002227  {$ENDC}
002228
002229  {$S SgTxtIni}
002230
002231
002232

End of File -- Lines: 2232 Characters: 80799
```

```
================================================================================================
FILE: "LIBTK/UTEXT4.TEXT"
================================================================================================

000001  {UText4}
000002                                      {TEXT SELECTION TYPES AND COMMANDS}
000003
000004
000005  {changed 04/27/84 1307  Change to TTypingCmd.Perform}
000006  {changed 04/25/84 1406  Got rid of IF numParas = 1 in Adjust proc in InsertText}
000007  {changed 04/25/84 1250  Changed TStyleCmd.FilterAndDo back to filtering TParaImage for Compugraphic}
000008  {changed 04/19/84 1308  Set up fields for partial line erasing in KeyPause}
000009  {changed 04/17/84 1720  In TTextSelection.CutCopy, changed "IF firstLP >= 0" to "IF firstLP > 0"}
000010  {changed 04/16/84 1414  Set viewLRect origined at (0,0) in TTextSelection.CutCopy}
000011  {changed 04/16/84 1322  Set textImage.minHeight to 0 in TOnePara and TMultiPara.CopySelf}
000012  {changed 04/16/84 1024  Call recomputeImages in TOnePara and TMultiPara.CopySelf to set clipView size}
000013  {changed 04/13/84 1736  First parameter decremented in calls to TParagraph.StyleAt;
000014                          No longer set unHighlightBefore[doPhase] to FALSE in typingCmd;
000015                          Set TInsertionPoint.justReturned to FALSE in all Key methods (except KeyReturn);
000016                          Call StyleFromContext in TStyleCmd.Perform}
000017  {changed 04/13/84 1531  TStyleCmd.FilterAndDo now filters a TEditPara rather than a TParaImage;
000018                          Call StyleFromContext through a filter in TTextSelection.CREATE}
000019  {changed 04/13/84 1102  TInsertion.MousePress: Allow selecting of word when double click on last half
000020                              of last character in word}
000021  {changed 04/13/84 0209  Changed all calls to TEditPara.CREATE to TTextImage.NewEditPara}
000022  {changed 04/09/84 1530  Set highlightAfter[doPhase] FALSE in TTypingCmd.CREATE}
000023  {changed 04/09/84 1337  Set deferUpdate TRUE in TTypingCmd.Perform (do phase)}
000024  {changed 04/09/84 1202  TInsertion.MousePress: Made sure all insertion points had their isPara flags
000025                              set when triple click on an empty paragraph;
000026                          TInsertionPoint.MouseMove: Don't unHighlight if isPara was TRUE}
000027
000028
000029
000030  {$S SgTxtHot}
000031
000032
000033  METHODS OF TTextSelection;
000034
000035  {$S SgTxtHot}
000036      FUNCTION TTextSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
000037                                                  itsTextImage: TTextImage; itsAnchorLPt: LPoint;
000038                                                  beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
000039                                                  endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER
000040                                                  ): TTextSelection;
000041
000042          {Need to filter paragraph before asking about its type styles}
000043          PROCEDURE FindFilteredStyle(obj: TObject);
```

```
000044              BEGIN
000045              SELF.StyleFromContext;
000046              END;
000047
000048      VAR range:  TTextRange;
000049      BEGIN
000050          {$IFC fTrace}BP(9);{$ENDC}
000051          IF object = NIL THEN
000052              object := NewObject(heap, THISCLASS);
000053          SELF := TTextSelection(TSelection.CREATE(object, heap, itsView, somethingKind, itsAnchorLPt));
000054
000055          range := TTextRange.CREATE(NIL, heap, beginPara, beginIndex, beginLP, endPara, endIndex, endLP);
000056          WITH SELF DO
000057              BEGIN
000058              textImage := itsTextImage;
000059              textRange := range;
000060              isWordSelection := FALSE;
000061              isParaSelection := FALSE;
000062              amTyping := FALSE;
000063              viewTick := -1; { force recalculation of selection range }
000064              END;
000065          SELF.textImage.FilterAndDo(TParaImage(beginPara.images.First), FindFilteredStyle);
000066          {$IFC fTrace}EP;{$ENDC}
000067      END;
000068
000069
000070  {$S SgTxtHot}
000071      PROCEDURE TTextSelection.Free;
000072      BEGIN
000073          {$IFC fTrace}BP(10);{$ENDC}
000074          SELF.textRange.Free;
000075          SUPERSELF.Free;
000076          {$IFC fTrace}EP;{$ENDC}
000077      END;
000078
000079
000080  {$S SgTxtWrm}
000081      FUNCTION TTextSelection.Clone(heap: THeap): TObject;
000082      VAR textSel:    TTextSelection;
000083          range:      TTextRange;
000084      BEGIN
000085          {$IFC fTrace}BP(10);{$ENDC}
000086          range := TTextRange(SELF.textRange.Clone(heap));
000087          textSel := TTextSelection(SUPERSELF.Clone(heap));
000088          textSel.textRange := range;
000089          Clone := textSel;
000090          {$IFC fTrace}EP;{$ENDC}
000091      END;
```

```
000092
000093
000094  {$S SgTxtCld}
000095  {$IFC fTextTrace}
000096      PROCEDURE TTextSelection.Fields(PROCEDURE Field(nameAndType: S255));
000097      BEGIN
000098          SUPERSELF.Fields(Field);
000099          Field('textImage: TTextImage');
000100          Field('textRange: TTextRange');
000101          Field('isWordSelection: BOOLEAN');
000102          Field('isParaSelection: BOOLEAN');
000103          Field('viewTick: INTEGER');
000104          Field('amTyping: BOOLEAN');
000105          Field('onFaces: INTEGER');
000106          Field('font: INTEGER');
000107          Field('');
000108      END;
000109  {$ENDC}
000110
000111
000112  {$S SgTxtHot}
000113      FUNCTION TTextSelection.BecomeInsertionPoint: TInsertionPoint;
000114      VAR insertionPt:    TInsertionPoint;
000115      BEGIN
000116          {$IFC fTrace}BP(11);{$ENDC}
000117          insertionPt := TInsertionPoint(SELF.FreedAndReplacedBy(
000118                                  TInsertionPoint.CREATE(NIL, SELF.Heap,
000119                                                         SELF.view, SELF.textImage, SELF.anchorLPt,
000120                                                             SELF.textRange.firstPara,
000121                                                             SELF.textRange.firstIndex,
000122                                                             SELF.textRange.firstLP)));
000123          SELF.textImage.text.ChangeSelInOtherPanels(insertionPt);
000124          BecomeInsertionPoint := insertionPt;
000125          {$IFC fTrace}EP;{$ENDC}
000126      END;
000127
000128
000129  {$S SgTxtWrm}
000130      FUNCTION TTextSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
000131      VAR typeStyle:  TTypeStyle;
000132      BEGIN
000133          {$IFC fTrace}BP(10);{$ENDC}
000134          CASE cmdNumber OF
000135              uModern, uClassic:
000136                  BEGIN
000137                  CanDoCommand := TRUE;
000138                  typeStyle := SELF.currTypeStyle;
000139                  checkIt := (cmdNumber - uModern + 1) = typeStyle.font.fontFamily;
```

```
000140              END;
000141
000142          u20Pitch, u15Pitch, u12Pitch, u10Pitch, u12Point, u14Point, u18Point, u24Point:
000143              BEGIN
000144              CanDoCommand := TRUE;
000145              typeStyle := SELF.currTypeStyle;
000146              checkIt := (cmdNumber - u20Pitch + 1) = typeStyle.font.fontSize;
000147              END;
000148
000149          {$IFC LibraryVersion <= 20}
000150          uFnt0, uFnt1, uFnt2, uFnt3, uFnt4, uFnt5, uFnt6, uFnt7, uFnt8, uFnt9, uFnt10, uFnt11:
000151              BEGIN
000152              CanDoCommand := TRUE;
000153              WITH SELF.currTypeStyle.font DO
000154                  CASE cmdNumber OF
000155                      uFnt0:   checkIt := fontFamily = famSystem;
000156                      uFnt1:   checkIt := (fontFamily = famModern)  AND (fontSize = size15Pitch);
000157                      uFnt2:   checkIt := (fontFamily = famModern)  AND (fontSize = size12Pitch);
000158                      uFnt3:   checkIt := (fontFamily = famClassic) AND (fontSize = size12Pitch);
000159                      uFnt4:   checkIt := (fontFamily = famModern)  AND (fontSize = size10Pitch);
000160                      uFnt5:   checkIt := (fontFamily = famModern)  AND (fontSize = size14Point);
000161                      uFnt6:   checkIt := (fontFamily = famModern)  AND (fontSize = size12Point);
000162                      uFnt7:   checkIt := (fontFamily = famClassic) AND (fontSize = size12Point);
000163                      uFnt8:   checkIt := (fontFamily = famModern)  AND (fontSize = size18Point);
000164                      uFnt9:   checkIt := (fontFamily = famClassic) AND (fontSize = size18Point);
000165                      uFnt10:  checkIt := (fontFamily = famModern)  AND (fontSize = size24Point);
000166                      uFnt11:  checkIt := (fontFamily = famClassic) AND (fontSize = size24Point);
000167                      END;
000168              END;
000169          {$ENDC}
000170
000171          uPlain, uBold, uItalic, uUnderline, uShadow, uOutline:
000172              BEGIN
000173              CanDoCommand := TRUE;
000174              typeStyle := SELF.currTypeStyle;
000175              WITH typeStyle DO
000176                  CASE cmdNumber OF
000177                      uPlain:     checkIt := onFaces=[];
000178                      uBold:      checkIt := (bold       in onFaces);
000179                      uItalic:    checkIt := (italic     in onFaces);
000180                      uUnderline: checkIt := (underlined in onFaces);
000181                      uShadow:    checkIt := (shadow     in onFaces);
000182                      uOutline:   checkIt := (outline    in onFaces);
000183                      END;
000184              END;
000185
000186          uCut, uCopy, uClear:
000187              CanDoCommand := TRUE;
```

```
000188
000189              uPaste:
000190                  BEGIN
000191                  clipboard.Inspect;
000192                  IF (clipboard.hasView) OR (clipBoard.hasUniversalText) THEN
000193                      CanDoCommand := TRUE;
000194                  END;
000195
000196              OTHERWISE
000197                  CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
000198              END;
000199          {$IFC fTrace}EP;{$ENDC}
000200      END;
000201
000202
000203  {$S SgTxtCld}
000204      PROCEDURE TTextSelection.ChangeStyle(cmdNumber: TCmdNumber);
000205      BEGIN
000206          {$IFC fTrace}BP(9);{$ENDC}
000207          Write(chr(7),'Change typestyle not implemented for this selection type. ');
000208          WriteLn ('(cmdNumber= ',cmdNumber:1,')');
000209          {$IFC fTrace}EP;{$ENDC}
000210      END;
000211
000212
000213  {$S SgTxtHot}
000214      PROCEDURE TTextSelection.ChangeText(PROCEDURE TextEdit; PROCEDURE Adjust);
000215      BEGIN
000216          {$IFC fTrace}BP(10);{$ENDC}
000217          SELF.MarkChanged;
000218          TextEdit;
000219          Adjust;
000220          IF NOT deferUpdate THEN
000221              BEGIN
000222              SELF.textImage.text.RecomputeImages;
000223              (*SELF.textImage.text.Invalidate;*)
000224              END;
000225          {$IFC fTrace}EP;{$ENDC}
000226      END;
000227
000228
000229  {$S SgTxtCld}
000230      FUNCTION TTextSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection;
000231      BEGIN
000232          {$IFC fTrace}BP(11);{$ENDC}
000233          CopySelf := NIL;
000234          {$IFC fTrace}EP;{$ENDC}
000235      END;
```

```
000236
000237
000238  {$S SgTxtCld}
000239      PROCEDURE TTextSelection.CutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN);
000240      VAR clipHeap:           THeap;
000241          clipPanel:          TPanel;
000242          clipView:           TTextView;
000243          viewLRect:          LRect;
000244          text:               TText;
000245          insPt:              TInsertionPoint;
000246          clipTextSelection:  TMultiParaSelection;
000247          selSize:            INTEGER;
000248          checkLast:          BOOLEAN;
000249
000250      BEGIN
000251          {$IFC fTrace}BP(11);{$ENDC}
000252          clipHeap := clipSelection.heap;
000253          clipPanel := clipSelection.panel;
000254          viewLRect := SELF.textImage.extentLRect;
000255          SetLRect(viewLRect, 0, 0, viewLRect.right - viewLRect.left + 2 * cHorizMargin,
000256                                    viewLRect.bottom - viewLRect.top + 2 * cVertMargin);
000257          clipView := TTextView.CREATE(NIL, clipHeap, clipPanel, viewLRect);
000258          clipTextSelection := TMultiParaSelection(clipSelection.FreedAndReplacedBy(
000259                                                          SELF.CopySelf(clipHeap, clipView)));
000260          clipView.valid := TRUE;
000261          clipTextSelection.isParaSelection := SELF.isParaSelection;
000262          clipTextSelection.isWordSelection := SELF.isWordSelection;
000263          {Intelligence!!  Even if word flag not set, let's see if we qualify as a word selection anyway}
000264          IF NOT SELF.isWordSelection THEN
000265              WITH SELF.textRange DO
000266                  {$H-}
000267                  clipTextSelection.isWordSelection := firstPara.Qualifies(firstLP) AND
000268                                                       lastPara.Qualifies(lastLP-1) AND
000269                                                       NOT firstPara.Qualifies(firstLP-1) AND
000270                                                       NOT lastPara.Qualifies(lastLP);
000271                  {$H+}
000272          clipView.textImage := clipTextSelection.textImage;
000273          IF deleteOriginal THEN
000274              BEGIN
000275              {Intelligent Cut: Delete extra space}
000276              IF clipTextSelection.isWordSelection THEN
000277                  WITH SELF.textRange DO
000278                      BEGIN
000279                      {$H-}
000280                      checkLast := TRUE;
000281                      IF firstLP > 0 THEN
000282                          IF firstPara.At(firstLp) = ' ' THEN
000283                              BEGIN
```

```
000284                          firstLP := firstLP - 1;
000285                          checkLast := FALSE;
000286                          END;
000287                      IF checkLast AND (lastLP < lastPara.size) THEN
000288                          IF lastPara.At(lastLP + 1) = ' ' THEN
000289                          lastLP := lastLP + 1;
000290                      {$H+}
000291                      END;
000292              SELF.DeleteAndFree;
000293              insPt := SELF.BecomeInsertionPoint;
000294              END;
000295          {$IFC fTrace}EP;{$ENDC}
000296      END;
000297
000298
000299  {$S SgTxtCld}
000300      PROCEDURE TTextSelection.DeleteAndFree;
000301      BEGIN
000302          {$IFC fTrace}BP(9);{$ENDC}
000303          {$IFC fTrace}EP;{$ENDC}
000304      END;
000305
000306
000307  {$S SgTxtCld}
000308      FUNCTION TTextSelection.DeleteButSave: TText;
000309      BEGIN
000310          {$IFC fTrace}BP(9);{$ENDC}
000311          DeleteButSave := NIL;
000312          {$IFC fTrace}EP;{$ENDC}
000313      END;
000314
000315
000316  {$S SgTxtCld}
000317      PROCEDURE TTextSelection.DoChangeStyle(cmdNumber: TCmdNumber; paragraph: TParagraph;
000318                                             firstLP: INTEGER; lastLP: INTEGER; VAR newStyle: TTypeStyle);
000319      VAR onFaces:    {$IFC LibraryVersion <= 20}TSeteface{$ELSEC}Style{$ENDC};
000320          faceChange: BOOLEAN;
000321      BEGIN
000322          {$IFC fTrace}BP(10);{$ENDC}
000323          {$IFC fTextTrace}
000324          IF fTextTrace THEN
000325              BEGIN
000326              WriteLn('_+_ Entering TTextSelection.DoChangeStyle: firstLP = ', firstLP:1,
000327                                                                  '  lastLP = ', lastLP:1);
000328              END;
000329          {$ENDC}
000330          faceChange := TRUE;
000331          CASE cmdNumber OF
```

```
000332              uPlain:
000333                  onFaces := [];
000334              uBold:
000335                  onFaces := [bold];
000336              uItalic:
000337                  onFaces := [italic];
000338              uUnderline:
000339                  onFaces := [underlined];
000340              uShadow:
000341                  onFaces := [shadow];
000342              uOutline:
000343                  onFaces := [outline];
000344
000345              uModern, uClassic:
000346                  BEGIN
000347                  faceChange := FALSE;
000348                  paragraph.ChgFontFamily(firstLP, lastLP, cmdNumber - uModern + 1, newStyle);
000349                  END;
000350
000351              {$IFC LibraryVersion <= 20}
000352              uFnt0, uFnt1, uFnt2, uFnt3, uFnt4, uFnt5, uFnt6, uFnt7, uFnt8, uFnt9, uFnt10, uFnt11:
000353                  BEGIN
000354                  faceChange := FALSE;
000355                  CASE cmdNumber OF
000356                      uFnt0:   paragraph.ChgFontFamily(firstLP, lastLP, famSystem, newStyle);
000357                      uFnt1:
000358                          BEGIN
000359                          paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
000360                          paragraph.ChgFontSize(firstLP, lastLP, size15Pitch, newStyle);
000361                          END;
000362                      uFnt2:
000363                          BEGIN
000364                          paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
000365                          paragraph.ChgFontSize(firstLP, lastLP, size12Pitch, newStyle);
000366                          END;
000367                     uFnt3:
000368                          BEGIN
000369                          paragraph.ChgFontFamily(firstLP, lastLP, famClassic, newStyle);
000370                          paragraph.ChgFontSize(firstLP, lastLP, size12Pitch, newStyle);
000371                          END;
000372                     uFnt4:
000373                          BEGIN
000374                          paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
000375                          paragraph.ChgFontSize(firstLP, lastLP, size10Pitch, newStyle);
000376                          END;
000377                      uFnt5:
000378                          BEGIN
000379                          paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
```

```
000380                      paragraph.ChgFontSize(firstLP, lastLP, size14Point, newStyle);
000381                      END;
000382                  uFnt6:
000383                      BEGIN
000384                      paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
000385                      paragraph.ChgFontSize(firstLP, lastLP, size12Point, newStyle);
000386                      END;
000387                  uFnt7:
000388                      BEGIN
000389                      paragraph.ChgFontFamily(firstLP, lastLP, famClassic, newStyle);
000390                      paragraph.ChgFontSize(firstLP, lastLP, size12Point, newStyle);
000391                      END;
000392                  uFnt8:
000393                      BEGIN
000394                      paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
000395                      paragraph.ChgFontSize(firstLP, lastLP, size18Point, newStyle);
000396                      END;
000397                  uFnt9:
000398                      BEGIN
000399                      paragraph.ChgFontFamily(firstLP, lastLP, famClassic, newStyle);
000400                      paragraph.ChgFontSize(firstLP, lastLP, size18Point, newStyle);
000401                      END;
000402                  uFnt10:
000403                      BEGIN
000404                      paragraph.ChgFontFamily(firstLP, lastLP, famModern, newStyle);
000405                      paragraph.ChgFontSize(firstLP, lastLP, size24Point, newStyle);
000406                      END;
000407                  uFnt11:
000408                      BEGIN
000409                      paragraph.ChgFontFamily(firstLP, lastLP, famClassic, newStyle);
000410                      paragraph.ChgFontSize(firstLP, lastLP, size24Point, newStyle);
000411                      END;
000412                  END;
000413              END;
000414          {$ENDC}
000415
000416          OTHERWISE
000417              BEGIN
000418              faceChange := FALSE;
000419              paragraph.ChgFontSize(firstLP, lastLP, cmdNumber - u20Pitch + 1, newStyle);
000420              END;
000421          END;
000422
000423      IF faceChange THEN
000424          paragraph.ChgFace(firstLP, lastLP, onFaces, newStyle);
000425      {$IFC fTrace}EP;{$ENDC}
000426  END;
000427
```

```
000428
000429  {$S SgTxtCld}
000430      PROCEDURE TTextSelection.GetHysteresis(VAR hysterPt: Point);
000431      BEGIN
000432          {$IFC fTrace}BP(6);{$ENDC}
000433          hysterPt := zeroPt;
000434          {$IFC fTrace}EP;{$ENDC}
000435      END;
000436
000437
000438  {$S SgTxtHot}
000439      PROCEDURE TTextSelection.Highlight(highTransit: THighTransit);
000440      BEGIN
000441          {$IFC fTrace}BP(10);{$ENDC}
000442          {Note that this is called from TPanel.Highlight which does an OnAllPadsDo, thus we do not
000443              call TText.HiliteParagraphs, but rather we call TTextImage.HiliteText }
000444          IF SELF.textImage.imageList.size > 0 THEN
000445              WITH SELF.textRange DO
000446                  {$H-}
000447                  SELF.textImage.headTxtImg.HiliteText(highTransit, firstIndex, firstLP, lastIndex, lastLP,
000448                                                       SELF.isParaSelection);
000449                  {$H+}
000450          {$IFC fTrace}EP;{$ENDC}
000451      END;
000452
000453
000454  {$S SgTxtCld}
000455      PROCEDURE TTextSelection.Invalidate;
000456      BEGIN
000457          {$IFC fTrace}BP(10);{$ENDC}
000458          SELF.textImage.text.Invalidate;
000459          {$IFC fTrace}EP;{$ENDC}
000460      END;
000461
000462
000463  (* This is now done automatically by the ToolKit
000464  {$S SgTxtHot}
000465      PROCEDURE TTextSelection.KeyClear;
000466      BEGIN
000467          {$IFC fTrace}BP(10);{$ENDC}
000468          SELF.window.PerformCommand(TClearTextCmd.CREATE(NIL, SELF.Heap, uClear,
000469                                                       SELF.textImage, SELF.textImage.text));
000470          {$IFC fTrace}EP;{$ENDC}
000471      END;
000472  *)
000473
000474
000475  {$S SgTxtHot}
```

```
000476      PROCEDURE TTextSelection.KeyBack(fWord: BOOLEAN);
000477      BEGIN
000478          {$IFC fTrace}BP(10);{$ENDC}
000479          {This will create the typing command, which deletes any selected text}
000480          SELF.KeyText;
000481          {$IFC fTrace}EP;{$ENDC}
000482      END;
000483
000484
000485  {$S SgTxtHot}
000486      PROCEDURE TTextSelection.KeyChar(ch: CHAR);
000487      BEGIN
000488          {$IFC fTrace}BP(10);{$ENDC}
000489          SELF.KeyText;
000490          SELF.KeyChar(ch); {!!! Assumes SELF was converted to insertion point}
000491          {$IFC fTrace}EP;{$ENDC}
000492      END;
000493
000494
000495  {$S SgTxtWrm}
000496      PROCEDURE TTextSelection.KeyReturn;
000497      BEGIN
000498          {$IFC fTrace}BP(10);{$ENDC}
000499          SELF.KeyText;
000500          SELF.KeyReturn; {!!! Assumes SELF was converted to insertion point}
000501          {$IFC fTrace}EP;{$ENDC}
000502      END;
000503
000504
000505  {$S SgTxtHot}
000506      PROCEDURE TTextSelection.KeyText;
000507      BEGIN
000508          {$IFC fTrace}BP(7);{$ENDC}
000509          IF NOT SELF.amTyping THEN
000510              SELF.window.PerformCommand(TTypingCmd.CREATE(NIL, SELF.Heap,
000511                                                  SELF.textImage, SELF.textImage.text));
000512          {$IFC fTrace}EP;{$ENDC}
000513      END;
000514
000515
000516  {$S SgTxtHot}
000517      PROCEDURE TTextSelection.MarkChanged;
000518      BEGIN
000519          {$IFC fTrace}BP(10);{$ENDC}
000520          SUPERSELF.MarkChanged;
000521          SELF.textImage.text.MarkChanged(SELF.textRange);
000522          {$IFC fTrace}EP;{$ENDC}
000523      END;
```

```
000524
000525
000526  {$S SgTxtCld}
000527      PROCEDURE TTextSelection.MousePress(mouseLPt: LPoint);
000528      BEGIN
000529          {$IFC fTrace}BP(10);{$ENDC}
000530          IF clickState.fShift THEN
000531              SELF.MouseMove(mouseLPt)
000532          ELSE
000533              SELF.textImage.MousePress(mouseLPt);
000534          {$IFC fTrace}EP;{$ENDC}
000535      END;
000536
000537
000538  {$S SgTxtHot}
000539      PROCEDURE TTextSelection.MouseRelease;
000540      BEGIN
000541          {$IFC fTrace}BP(10);{$ENDC}
000542          ObscureCursor;
000543          {$IFC fTrace}EP;{$ENDC}
000544      END;
000545
000546
000547  {$S SgTxtCld}
000548      FUNCTION TTextSelection.NewCommand(cmdNumber: TCmdNumber): TCommand;
000549      VAR heap:           THeap;
000550          textImage:      TTextImage;
000551      BEGIN
000552          {$IFC fTrace}BP(10);{$ENDC}
000553          NewCommand := NIL;
000554          heap := SELF.heap;
000555          textImage := SELF.textImage;
000556          CASE cmdNumber OF
000557              u20Pitch, u15Pitch, u12Pitch, u10Pitch, u12Point, u14Point, u18Point, u24Point,
000558                      uModern, uClassic,
000559                      uPlain, uBold, uItalic, uUnderline, uShadow, uOutline:
000560                  NewCommand := SELF.NewStyleCmd(heap, cmdNumber, textImage);
000561
000562              {$IFC LibraryVersion <= 20}
000563              uFnt0, uFnt1, uFnt2, uFnt3, uFnt4, uFnt5, uFnt6, uFnt7, uFnt8, uFnt9, uFnt10, uFnt11:
000564                  NewCommand := SELF.NewStyleCmd(heap, cmdNumber, textImage);
000565              {$ENDC}
000566
000567              uCut, uCopy:
000568                  NewCommand := SELF.NewCutCopyCmd(heap, cmdNumber, textImage);
000569
000570              uClear:
000571                  NewCommand := TClearTextCmd.CREATE(NIL, SELF.Heap, uClear, textImage, textImage.text);
```

```
000572
000573              uPaste:
000574                  NewCommand := TTextPaste.CREATE(NIL, heap, textImage, textImage.text);
000575
000576              {cFormatChanges:}
000577
000578              OTHERWISE
000579                  NewCommand := SUPERSELF.NewCommand(cmdNumber);
000580              END;
000581          {$IFC fTrace}EP;{$ENDC}
000582      END;
000583
000584
000585  {$S SgTxtCld}
000586      FUNCTION TTextSelection.NewCutCopyCmd(heap: THeap; cmdNumber: TCmdNumber;
000587                                      textImage: TTextImage): TCommand;
000588      BEGIN
000589          {$IFC fTrace}BP(10);{$ENDC}
000590          NewCutCopyCmd := TTextCutCopy.CREATE(NIL, heap, cmdNumber, textImage,
000591                                                  cmdNumber=uCut, textImage.text);
000592          {$IFC fTrace}EP;{$ENDC}
000593      END;
000594
000595
000596  {$S SgTxtCld}
000597      FUNCTION TTextSelection.NewStyleCmd(heap: THeap; cmdNumber: TCmdNumber;
000598                                      textImage: TTextImage): TCommand;
000599      BEGIN
000600          {$IFC fTrace}BP(10);{$ENDC}
000601          WITH SELF.textRange DO
000602              {$H-}
000603              NewStyleCmd := TStyleCmd.CREATE(NIL, heap, cmdNumber, textImage, firstIndex, lastIndex,
000604                                                  firstLP, lastLP, SELF);
000605              {$H+}
000606          {$IFC fTrace}EP;{$ENDC}
000607      END;
000608
000609
000610  {$S SgTxtCld}
000611      FUNCTION TTextSelection.ReplicateForOtherPanel(itsTextImage: TTextImage): TTextSelection;
000612      VAR sel:    TTextSelection;
000613      BEGIN
000614          {$IFC fTrace}BP(9);{$ENDC}
000615          WITH SELF.textRange DO
000616              {$H-}
000617              sel := itsTextImage.NewTextSelection(firstPara, firstIndex, firstLP,
000618                                                   lastPara, lastIndex, lastLP);
000619              {$H+}
```

```
000620          sel.isParaSelection := SELF.isParaSelection;
000621          sel.isWordSelection := SELF.isWordSelection;
000622          ReplicateForOtherPanel := sel;
000623          {$IFC fTrace}EP;{$ENDC}
000624      END;
000625
000626
000627  {$S SgTxtCld}
000628      PROCEDURE TTextSelection.StyleFromContext;
000629      BEGIN
000630          {$IFC fTrace}BP(9);{$ENDC}
000631          {$IFC fTrace}EP;{$ENDC}
000632      END;
000633
000634
000635  {$S SgTxtIni}
000636  END; {METHODS OF TTextSelection}
000637
000638
000639  METHODS OF TInsertionPoint;
000640
000641  {$S SgTxtHot}
000642      FUNCTION TInsertionPoint.CREATE(object: TObject; heap: THeap; itsView: TView; itsTextImage: TTextImage;
000643                                      itsAnchorLPt: LPoint; itsParagraph: TEditPara; itsIndex: LONGINT;
000644                                      itsLP: INTEGER): TInsertionPoint;
000645      VAR s:          TListScanner;
000646          paragraph:  TEditPara;
000647      BEGIN
000648          {$IFC fTrace}BP(10);{$ENDC}
000649          IF object = NIL THEN
000650              object := NewObject(heap, THISCLASS);
000651
000652          IF itsIndex <= 0 THEN
000653              BEGIN
000654              s := itsTextImage.text.paragraphs.Scanner;
000655              WHILE s.Scan(paragraph) DO
000656                  IF paragraph = itsParagraph THEN
000657                      BEGIN
000658                      itsIndex := s.position;
000659                      s.Done;
000660                      END;
000661              END;
000662
000663          SELF := TInsertionPoint(TTextSelection.CREATE(object, heap, itsView, itsTextImage, itsAnchorLPt,
000664                                                        itsParagraph, itsIndex, itsLP, itsParagraph, itsIndex, itsLP));
000665
000666          itsParagraph.bsCount := 0;
000667          WITH SELF DO
```

```
000668              BEGIN
000669              typingCmd := NIL;
000670              styleCmdNumber := 0;
000671              newestLP := itsLP;
000672              nextHighTransit := hOnToOff;
000673              justReturned := FALSE;
000674              END;
000675          {$IFC fTrace}EP;{$ENDC}
000676      END;
000677
000678
000679  {$S SgTxtHot}
000680      PROCEDURE TInsertionPoint.Free;
000681      BEGIN
000682          {$IFC fTrace}BP(10);{$ENDC}
000683          IF SELF.typingCmd <> NIL THEN
000684              WITH SELF, typingCmd.typingRange DO
000685                  BEGIN
000686                  lastPara := textrange.firstPara;
000687                  lastIndex := textrange.firstIndex;
000688                  lastLP := textrange.firstLP;
000689                  END;
000690          (*
000691          SELF.textRange.firstPara.StopEdit;
000692          *)
000693          SUPERSELF.Free;
000694          {$IFC fTrace}EP;{$ENDC}
000695      END;
000696
000697
000698  {$S SgTxtWrm}
000699      FUNCTION TInsertionPoint.Clone(heap: THeap): TObject;
000700      VAR insPt:  TInsertionPoint;
000701      BEGIN
000702          {$IFC fTrace}BP(10);{$ENDC}
000703          insPt := TInsertionPoint(SUPERSELF.Clone(heap));
000704          insPt.typingCmd := NIL;
000705          insPt.amTyping := FALSE;
000706          Clone := insPt;
000707          {$IFC fTrace}EP;{$ENDC}
000708      END;
000709
000710
000711
000712  {$S SgTxtCld}
000713  {$IFC fTextTrace}
000714      PROCEDURE TInsertionPoint.Fields(PROCEDURE Field(nameAndType: S255));
000715      BEGIN
```

```
000716          TTextSelection.Fields(Field);
000717          Field('typingCmd: TTypingCmd');
000718          Field('styleCmdNumber: INTEGER');
000719          Field('newestLP: INTEGER');
000720          Field('justReturned: BOOLEAN');
000721          Field('nextHighTransit: Byte');
000722          Field('nextTransitTime: LONGINT');
000723          Field('');
000724      END;
000725  {$ENDC}
000726
000727
000728  {$S SgTxtHot}
000729      FUNCTION TInsertionPoint.BecomeInsertionPoint: TInsertionPoint;
000730      BEGIN
000731          {$IFC fTrace}BP(11);{$ENDC}
000732          BecomeInsertionPoint := SELF;
000733          {$IFC fTrace}EP;{$ENDC}
000734      END;
000735
000736
000737  {$S SgTxtWrm}
000738      FUNCTION TInsertionPoint.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
000739      BEGIN
000740          {$IFC fTrace}BP(10);{$ENDC}
000741          CASE cmdNumber OF
000742              uCut, uCopy:
000743                  CanDoCommand := FALSE;
000744
000745              OTHERWISE
000746                  CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
000747              END;
000748          {$IFC fTrace}EP;{$ENDC}
000749      END;
000750
000751
000752  {$S SgTxtCld}
000753      PROCEDURE TInsertionPoint.CutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN);
000754      BEGIN
000755          {$IFC fTrace}BP(11);{$ENDC}
000756          (* Staged Alert *)
000757          {$IFC fTrace}EP;{$ENDC}
000758      END;
000759
000760
000761  {$S SgTxtCld}
000762      PROCEDURE TInsertionPoint.FinishPaste(clipSelection: TSelection; pic: PicHandle);
000763      VAR clipTextSelection:  TMultiParaSelection;
```

```
000764      BEGIN
000765          {$IFC fTrace}BP(11);{$ENDC}
000766          IF InClass(clipSelection, TMultiParaSelection) THEN
000767              BEGIN
000768              clipTextSelection := TMultiParaSelection(clipSelection);
000769              SELF.InsertText(clipTextSelection.textImage.text, clipTextSelection.isParaSelection,
000770                                                                clipTextSelection.isWordSelection, FALSE);
000771              END
000772          {$IFC fUseUnivText}
000773          ELSE IF clipBoard.hasUniversalText THEN
000774              SELF.InsertText(NIL, FALSE, FALSE, TRUE)
000775          {$ENDC};
000776
000777
000778          {$IFC fTrace}EP;{$ENDC}
000779      END;
000780
000781
000782  {$S SgTxtHot}
000783      PROCEDURE TInsertionPoint.IdleBegin(centiSeconds: LONGINT);
000784      {Assumes highlighting is already on}
000785      BEGIN
000786          {$IFC fTrace}BP(6);{$ENDC}
000787          IF (SELF.kind = nothingKind) OR SELF.isParaSelection THEN
000788              SELF.nextHighTransit := hNone
000789          ELSE
000790              BEGIN
000791              SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, FALSE);
000792              SELF.nextHighTransit := hOffTOOn;
000793              SELF.nextTransitTime := centiSeconds+blinkOffTime;
000794              END;
000795          SUPERSELF.IdleBegin(centiSeconds);
000796          {$IFC fTrace}EP;{$ENDC}
000797      END;
000798
000799
000800  {$S SgTxtHot}
000801      PROCEDURE TInsertionPoint.IdleContinue(centiSeconds: LONGINT);
000802      BEGIN
000803          {$IFC fTrace}BP(6);{$ENDC}
000804          IF (SELF.nextHighTransit<>hNone) AND (centiSeconds > SELF.nextTransitTime) THEN
000805              BEGIN
000806              SELF.textImage.text.HiliteRange(SELF.nextHighTransit, SELF.textRange, FALSE);
000807              WITH SELF DO
000808                  IF nextHighTransit = hOnToOff THEN
000809                      BEGIN
000810                      nextHighTransit := hOffToOn;
000811                      nextTransitTime := centiSeconds+blinkOffCentiSecs;
```

```
000812                  END
000813              ELSE
000814                  BEGIN
000815                  nextHighTransit := hOnToOff;
000816                  nextTransitTime := centiSeconds+blinkOnCentiSecs;
000817                  END;
000818              END;
000819          SUPERSELF.IdleContinue(centiSeconds);
000820          {$IFC fTrace}EP;{$ENDC}
000821      END;
000822
000823
000824  {$S SgTxtHot}
000825      PROCEDURE TInsertionPoint.IdleEnd(centiSeconds: LONGINT);
000826          { end with highlighting on }
000827      BEGIN
000828          {$IFC fTrace}BP(6);{$ENDC}
000829          IF SELF.nextHighTransit=hOffToOn THEN
000830              SELF.textImage.text.HiliteRange(hOffToOn, SELF.textRange, FALSE);
000831          SELF.nextHighTransit := hNone;
000832          SUPERSELF.IdleEnd(centiSeconds);
000833          {$IFC fTrace}EP;{$ENDC}
000834      END;
000835
000836
000837  {$S SgTxtCld}
000838      PROCEDURE TInsertionPoint.InsertText(text: TText; isParaSelection: BOOLEAN; isWordSelection: BOOLEAN;
000839                                                                                  universalText: BOOLEAN);
000840      VAR s:              TListScanner;
000841          prevPara:       TEditPara;
000842          newPara:        TEditPara;
000843          aParagraph:     TEditPara;
000844          newLP:          INTEGER;
000845          textImage:      TTextImage;
000846          insertIt:       BOOLEAN;
000847          done:           BOOLEAN;
000848          newParaImage:   TParaImage;
000849          paraIndex:      LONGINT;
000850          delta:          INTEGER;
000851          numParas:       INTEGER;
000852          needSpRight:    BOOLEAN;
000853          {$IFC fUseUnivText}
000854          readUnivText:   TTKReadUnivText;
000855          univPara:       TEditPara;
000856          univFormat:     TParaFormat;
000857          {$ENDC}
000858
000859          PROCEDURE StartPaste;
```

```
000860          BEGIN
000861              {$IFC fTrace}BP(10);{$ENDC}
000862              IF universalText THEN
000863                  BEGIN
000864                  {$IFC fUseUnivText}
000865                  univFormat := TParaFormat.CREATE(NIL, SELF.Heap, SELF.textImage.text.styleSheet);
000866                  univPara := textImage.NewEditPara(0, prevPara.format);
000867                  readUnivText := TTKReadUnivText.CREATE(NIL, SELF.Heap, NIL, 512,
000868                                                         [UTCharacters, UTParagraphs]);
000869                  numParas := 0;
000870                  {$ENDC}
000871                  END
000872              ELSE
000873                  BEGIN
000874                  numParas := text.paragraphs.size;
000875                  s := text.paragraphs.Scanner;
000876                  END;
000877              {$IFC fTrace}EP;{$ENDC}
000878          END;
000879
000880          PROCEDURE EndPaste;
000881          BEGIN
000882              {$IFC fTrace}BP(10);{$ENDC}
000883              IF universalText THEN
000884                  BEGIN
000885                  {$IFC fUseUnivText}
000886                  univPara.Free;
000887                  readUnivText.Free;
000888                  {$ENDC}
000889                  END;
000890              {$IFC fTrace}EP;{$ENDC}
000891          END;
000892
000893          FUNCTION GetParagraph(VAR paragraph: TEditPara): BOOLEAN;
000894          VAR currPos:        INTEGER;
000895              done:           BOOLEAN;
000896              runSize:        INTEGER;
000897              wasSomeText:    BOOLEAN;
000898              ch:             CHAR;
000899              typeStyle:      TTypeStyle;
000900          BEGIN
000901              {$IFC fTrace}BP(10);{$ENDC}
000902              If universalText THEN
000903                  BEGIN
000904                  {$IFC fUseUnivText}
000905                  univPara.ReplPString(0, univPara.Size, NIL);
000906                  currPos := 0;
000907                  wasSomeText := FALSE;
```

```
000908                  done := FALSE;
000909                  REPEAT
000910                      readUnivText.ReadRun;
000911                      runSize := readUnivText.data.size;
000912                      IF runSize > 0 THEN
000913                          BEGIN
000914                          IF NOT wasSomeText THEN
000915                              BEGIN
000916                              WITH univFormat, readUnivText.paragraphDescriptor DO
000917                                  BEGIN
000918                                  firstIndent := firstLineMargin;
000919                                  leftIndent := bodyMargin;
000920                                  (* Can't use this because it's given as distance from left rather than
000921                                     indent from right and I don't know what value of right edge of paper is.
000922                                  rightIndent := rightMargin;
000923                                  *)
000924                                  spaceBelowPara := paraLeading;
000925                                  END;
000926                              univPara.format := univFormat;
000927                              END;
000928                          wasSomeText := TRUE;
000929                          ch := readUnivText.data.At(runSize);
000930                          IF ORD(ch) = ascReturn THEN
000931                              BEGIN
000932                              readUnivText.data.DelAt(runSize);
000933                              runSize := runSize - 1;
000934                              numParas := numParas + 1;
000935                              done := TRUE;
000936                              END;
000937                          univPara.ReplTString(currPos, 0, readUnivText.data, 0, runSize);
000938                          typeStyle.onFaces := readUnivText.characterDescriptor.face;
000939                          typeStyle.font.fontFamily := uvFont[readUnivText.characterDescriptor.font].fontFamily;
000940                          typeStyle.font.fontSize := uvFont[readUnivText.characterDescriptor.font].fontSize;
000941                          univPara.NewStyle(currPos, currPos+runSize, typeStyle);
000942                          currPos := currPos + runSize;
000943                          END
000944                      ELSE
000945                          BEGIN
000946                          IF wasSomeText THEN
000947                              numParas := numParas + 1;
000948                          done := TRUE;
000949                          END;
000950                  UNTIL done;
000951                  IF wasSomeText THEN
000952                      paragraph := univPara
000953                  ELSE
000954                      paragraph := NIL;
000955                  GetParagraph := wasSomeText;
```

```
000956                  {$ELSEC}
000957                  paragraph := NIL;
000958                  GetParagraph := FALSE;
000959                  {$ENDC}
000960                  END
000961              ELSE
000962                  GetParagraph := s.Scan(paragraph);
000963              {$IFC fTrace}EP;{$ENDC}
000964          END;
000965
000966          PROCEDURE InsText;
000967          BEGIN
000968              {$IFC fTrace}BP(10);{$ENDC}
000969              delta := 0;
000970              textImage := SELF.textImage;
000971              newLP := SELF.textRange.firstLP;
000972              newPara := SELF.textRange.firstPara;
000973              prevPara := newPara;
000974              insertIt := FALSE;
000975
000976              IF isWordSelection THEN
000977                  BEGIN
000978                  needSpRight := newPara.Qualifies(newLP);
000979                  IF newPara.Qualifies(newLP-1) THEN
000980                      BEGIN
000981                      newPara.InsertOneChar(' ', newLP);
000982                      newLP := newLP + 1;
000983                      delta := 1;
000984                      END;
000985                  END;
000986
000987  (*          {special case: if first paragraph in text is designated a whole paragraph (by isParaSelection) AND
000988               if the insertion point (SELF) is at the end of the paragraph then we want to make a new
000989               paragraph rather than append it to the current paragraph and consequently set the flag that
000990               was supposed to prevent the first paragraph from being inserted}
000991              IF isParaSelection AND (prevPara.size = newLP) THEN
000992                  BEGIN
000993                  newPara := textImage.NewEditPara(0, prevPara.format);
000994                  newLP := 0;
000995                  insertIt := TRUE;
000996                  END;
000997  *)
000998              done := FALSE;
000999              StartPaste;
001000              IF GetParagraph(aParagraph) THEN
001001                  BEGIN
001002                  delta := delta + aParagraph.size;
001003                  REPEAT
```

```
001004                  newPara.ReplPara(newLP, 0, aParagraph, 0, aParagraph.size);
001005                  newLP := newLP + aParagraph.size;
001006                  IF insertIt THEN
001007                      textImage.text.InsParaAfter(prevPara, newPara);
001008                  insertIt := TRUE;
001009                  prevPara := newPara;
001010                  IF GetParagraph(aParagraph) THEN
001011                      BEGIN
001012                      newPara := textImage.NewEditPara(prevPara.size-newLP,
001013                                                       TParaFormat(aParagraph.format.Clone(SELF.Heap)));
001014                      {For now, so we don't get garbage (if aParagraph later deleted), put cloned
001015                       format on to styleSheet list}
001016                      SELF.textImage.text.styleSheet.formats.InsLast(newPara.format);
001017                      newPara.StartEdit(newPara.GrowSize);
001018                      newPara.ReplPara(0, 0, prevPara, newLp, prevPara.size - newLp);
001019                      prevPara.ReplPString(newLp, prevPara.size-newLP, NIL);
001020                      prevPara.StopEdit;
001021                      newLP := 0;
001022                      END
001023                   ELSE
001024                      done := TRUE;
001025                  UNTIL done;
001026                  END;
001027
001028              IF isParaSelection THEN
001029                  BEGIN
001030                  newPara := textImage.NewEditPara(prevPara.size - newLP, prevPara.format);
001031                  newPara.StartEdit(newPara.GrowSize);
001032                  newPara.ReplPara(0, 0, prevPara, newLp, prevPara.size - newLp);
001033                  prevPara.ReplPString(newLp, prevPara.size - newLP, NIL);
001034                  prevPara.StopEdit;
001035                  textImage.text.InsParaAfter(prevPara, newPara);
001036                  newPara := TEditPara(textImage.text.paragraphs.At(SELF.textRange.firstIndex + numParas));
001037                  numParas := numParas+1;
001038                  newLP := 0;
001039                  END
001040              ELSE IF isWordSelection THEN
001041                  IF needSpRight THEN
001042                      BEGIN
001043                      newPara.InsertOneChar(' ', newLP);
001044                      newLP := newLP + 1;
001045                      delta := delta + 1;
001046                      END;
001047
001048              EndPaste;
001049              {$IFC fTrace}EP;{$ENDC}
001050          END;
001051
```

```
001052          PROCEDURE Adjust;
001053              PROCEDURE AddDelta(paraImage: TParaImage);
001054              BEGIN
001055                  paraImage.AdjustLineLPs(SELF.textRange.firstLP, delta);
001056              END;
001057          BEGIN
001058              {$IFC fTrace}BP(10);{$ENDC}
001059              SELF.textRange.firstPara.EachImage(AddDelta);
001060
001061              WITH SELF, textRange DO
001062                  BEGIN
001063                  firstPara := newPara;
001064                  lastPara := newPara;
001065                  firstLP := newLP;
001066                  lastLP := newLP;
001067                  firstIndex := firstIndex + numParas - 1;
001068                  lastIndex := firstIndex;
001069                  newestLP := newLP;
001070                  amTyping := FALSE;
001071                  END;
001072              {$IFC fTrace}EP;{$ENDC}
001073          END;
001074      BEGIN
001075          {$IFC fTrace}BP(11);{$ENDC}
001076          IF (text <> NIL) OR universalText THEN
001077              SELF.ChangeText(InsText, Adjust);
001078          {$IFC fTrace}EP;{$ENDC}
001079      END;
001080
001081
001082  {$S SgTxtHot}
001083      PROCEDURE TInsertionPoint.KeyBack(fWord: BOOLEAN);
001084      VAR paragraph:      TEditPara;
001085          savedPara:      TEditPara;
001086          lp:             INTEGER;
001087          typingCmd:      TTypingCmd;
001088          prevPara:       TEditPara;
001089          textRange:      TTextRange;
001090
001091          PROCEDURE Adjust;
001092              VAR c:  CHAR;
001093
001094              PROCEDURE DelOne(paraImage: TParaImage);
001095              BEGIN
001096                  paraImage.AdjustLineLPs(lp, -1);
001097              END;
001098          BEGIN
001099              {$IFC fTrace}BP(10);{$ENDC}
```

```
001100              paragraph := SELF.textRange.firstPara;
001101              lp := SELF.newestLP;
001102
001103              IF fWord THEN
001104                  {&&& Move textRange.firstLP back to start of word};
001105
001106              paragraph.BeginInsertion(lp, 0);
001107
001108              IF (paragraph.holeStart < 1) AND (SELF.textRange.firstIndex = 1) THEN
001109                  SELF.CantDoIt
001110              ELSE
001111                  BEGIN
001112                  prevPara := NIL;
001113                  IF paragraph.holeStart < 1 THEN
001114                      BEGIN
001115                      {Backspacing over beginning of paragraph}
001116                      textRange := SELF.textRange;
001117                      textRange.firstIndex := textRange.firstIndex - 1;
001118                      prevPara := TEditPara(SELF.textImage.text.paragraphs.At(textRange.firstIndex));
001119                      WITH textRange DO
001120                          BEGIN
001121                          firstPara := prevPara;
001122                          firstLP := prevPara.size;
001123                          lastIndex := firstIndex;
001124                          lastPara := firstPara;
001125                          lastLP := firstLP;
001126                          SELF.newestLP := firstLP;
001127                          END;
001128                      prevPara.ReplPara(prevPara.size, 0, paragraph, 0, paragraph.size);
001129                      SELF.textImage.text.DelPara(paragraph, FALSE);
001130                      SELF.textImage.text.paragraphs.DelObject(paragraph, FALSE);
001131                      END
001132                  ELSE
001133                      BEGIN
001134                      paragraph.bsCount := paragraph.bsCount + 1;
001135                      c := paragraph.At(paragraph.holeStart);
001136                      paragraph.DelAt(paragraph.holeStart);
001137
001138                      paragraph.UpdateRuns(lp-1, 1, 0);
001139                      SELF.newestLP := lp-1;
001140                      END;
001141
001142                  typingCmd := SELF.typingCmd;
001143                  IF prevPara = NIL THEN
001144                      BEGIN
001145                      typingCmd.newCharCount := typingCmd.newCharCount - 1;
001146                      IF typingCmd.newCharCount < 0 THEN
001147                          BEGIN
```

```
001148                          typingCmd.typingRange.firstLP := typingCmd.typingRange.firstLP - 1;
001149                          typingCmd.newCharCount := 0;
001150                          savedPara := TEditPara(typingCmd.savedText.paragraphs.First);
001151                          {$R-}
001152                          savedPara.InsertOneChar(c, 0);
001153                          {$IFC fRngText}{$R+}{$ENDC}
001154                          END;
001155                      END
001156                  ELSE
001157                      BEGIN
001158                      typingCmd.newParaCount := typingCmd.newParaCount - 1;
001159                      IF typingCmd.newParaCount < 0 THEN
001160                          BEGIN
001161                          typingCmd.typingRange.firstIndex := textRange.firstIndex;
001162                          typingCmd.typingRange.firstPara := textRange.firstPara;
001163                          typingCmd.typingRange.firstLP := textRange.firstLP;
001164                          typingCmd.savedText.paragraphs.InsFirst(paragraph);
001165                          paragraph.ReplPString(0, paragraph.size, NIL);
001166                          typingCmd.newCharCount := 0;
001167                          typingCmd.newParaCount := 0;
001168                          END
001169                      ELSE
001170                          paragraph.Free;
001171                      END;
001172
001173
001174                  deferUpdate := TRUE;
001175                  SELF.justReturned := FALSE;
001176                  END;
001177              {$IFC fTrace}EP;{$ENDC}
001178          END;
001179
001180      BEGIN
001181          {$IFC fTrace}BP(10);{$ENDC}
001182          SELF.KeyText;
001183          Adjust;
001184          {$IFC fTrace}EP;{$ENDC}
001185      END;
001186
001187
001188  {$S SgTxtHot}
001189      PROCEDURE TInsertionPoint.KeyChar(ch: CHAR);
001190      BEGIN
001191          {$IFC fTrace}BP(10);{$ENDC}
001192          SELF.KeyText;
001193
001194          SELF.textRange.firstPara.InsertOneChar(ch, SELF.newestLP);
001195
```

```
001196          IF SELF.styleCmdNumber <> 0 THEN
001197              BEGIN
001198              SELF.textRange.firstPara.NewStyle(SELF.newestLP, SELF.newestLP+1, SELF.currTypeStyle);
001199              SELF.styleCmdNumber := 0;
001200              END;
001201          WITH SELF.typingCmd DO
001202              newCharCount := newCharCount + 1;
001203
001204          SELF.newestLP := SELF.newestLP + 1;
001205          SELF.justReturned := FALSE;
001206          {$IFC fTrace}EP;{$ENDC}
001207      END;
001208
001209
001210  {$S SgTxtCld}
001211      PROCEDURE TInsertionPoint.KeyEnter;
001212      BEGIN
001213          {$IFC fTrace}BP(10);{$ENDC}
001214          SELF.textImage.text.RecomputeImages;
001215          (*SELF.textImage.text.Invalidate;*)
001216          deferUpdate := FALSE;
001217          {$IFC fTrace}EP;{$ENDC}
001218      END;
001219
001220
001221  {$S SgTxtCld}
001222      PROCEDURE TInsertionPoint.KeyForward(fWord: BOOLEAN);
001223      VAR paragraph:      TEditPara;
001224          savedPara:      TEditPara;
001225          lp:             INTEGER;
001226
001227      {NOTE: This first stab at KeyForward does NOT properly restore typestyles !!}
001228          PROCEDURE Adjust;
001229
001230              PROCEDURE AddOne(paraImage: TParaImage);
001231              BEGIN
001232                  paraImage.AdjustLineLPs(lp, 1);
001233              END;
001234          BEGIN
001235              paragraph := SELF.textRange.firstPara;
001236              lp := SELF.newestLP;
001237
001238              IF (lp = paragraph.holeStart) AND (paragraph.bsCount > 0) THEN
001239                  BEGIN
001240
001241                  IF fWord THEN
001242                      {&&& Recover word};
001243
```

```
001244                  WITH paragraph DO
001245                      BEGIN
001246                      bsCount := paragraph.bsCount - 1;
001247                      holeStart := paragraph.holeStart + 1;
001248                      holeSize := paragraph.holeSize - 1;
001249                      size := size + 1;
001250                      END;
001251
001252
001253                  paragraph.UpdateRuns(lp-1, 0, 1);
001254                  SELF.newestLP := lp+1;
001255
001256                  WITH SELF.typingCmd DO
001257                      newCharCount := newCharCount + 1;
001258
001259                  deferUpdate := TRUE;
001260                  END;
001261              SELF.justReturned := FALSE;
001262          END;
001263
001264      BEGIN
001265          {$IFC fTrace}BP(10);{$ENDC}
001266          SELF.KeyText;
001267          Adjust;
001268          {$IFC fTrace}EP;{$ENDC}
001269      END;
001270
001271
001272  {$S SgTxtHot}
001273      PROCEDURE TInsertionPoint.KeyPause;
001274      {Called by ToolKit when there are no keystrokes pending}
001275      VAR text:       TText;
001276          diff:       INTEGER;
001277          lp:         INTEGER;
001278          textImage:  TTextImage;
001279          paraImage:  TParaImage;
001280          startLine:  INTEGER;
001281          startPixel: LONGINT;
001282
001283          PROCEDURE AddDiff(paraImage: TParaImage);
001284          BEGIN
001285              paraImage.AdjustLineLPs(lp, diff);
001286          END;
001287
001288          PROCEDURE AdjustOtherInsPts(obj: TObject);
001289          VAR insertPt:   TInsertionPoint;
001290          BEGIN
001291              insertPt := TInsertionPoint(obj);
```

```
001292              paraImage := insertPt.textImage.ImageWith(SELF.textRange.firstPara, lp);
001293              IF paraImage <> NIL THEN
001294                  BEGIN
001295                  paraImage.LocateLP(lp, startLine, startPixel);
001296                  insertPt.textImage.firstLinePixel := startPixel;
001297                  insertPt.textImage.useFirstPixel := TRUE;
001298                  END
001299              ELSE
001300                  insertPt.textImage.useFirstPixel := FALSE;
001301
001302              WITH SELF, insertPt.textRange DO
001303                  BEGIN
001304                  firstLP := newestLP;
001305                  lastLP := newestLP;
001306                  IF firstIndex <> textRange.firstIndex THEN
001307                      BEGIN
001308                      firstIndex := textRange.firstIndex;
001309                      lastIndex := textRange.lastIndex;
001310                      firstPara := textRange.firstPara;
001311                      lastPara := textRange.lastPara;
001312                      END;
001313                  END;
001314          END;
001315
001316          PROCEDURE HiliteOtherInsPts(obj: TObject);
001317          VAR insertPt:   TInsertionPoint;
001318          BEGIN
001319              insertPt := TInsertionPoint(obj);
001320              IF insertPt.view.OKToDrawIn(insertPt.textImage.extentLRect) THEN
001321                  insertPt.panel.Highlight(insertPt, hOffToOn);
001322          END;
001323      BEGIN
001324          {$IFC fTrace}BP(10);{$ENDC}
001325          IF SELF.amTyping AND NOT SELF.justReturned THEN
001326              BEGIN
001327              textImage := SELF.textImage;
001328              text := textImage.text;
001329              diff := SELF.newestLP - SELF.textRange.firstLP;
001330              lp := Min(SELF.textRange.firstLP, SELF.newestLP);
001331              SELF.MarkChanged;
001332              SELF.textRange.firstPara.EachImage(AddDiff);
001333
001334              {set up textImage fields for minimum rectangle erase of first update line}
001335              paraImage := textImage.ImageWith(SELF.textRange.firstPara, lp);
001336              IF paraImage <> NIL THEN
001337                  BEGIN
001338                  paraImage.LocateLP(lp, startLine, startPixel);
001339                  textImage.firstLinePixel := startPixel;
```

```
001340                  textImage.useFirstPixel := TRUE;
001341                  END
001342              ELSE
001343                  textImage.useFirstPixel := FALSE;
001344
001345              SELF.textRange.firstLP := SELF.newestLP;
001346              SELF.textRange.lastLP := SELF.newestLP;
001347              IF SELF.typingCmd <> NIL THEN
001348                  IF SELF.typingCmd.otherInsPts <> NIL THEN
001349                      SELF.typingCmd.otherInsPts.Each(AdjustOtherInsPts);
001350
001351              text.RecomputeImages;
001352
001353              {If view.OKToDrawIn was TRUE then we won't be going through the update cycle and hence the
001354               ToolKit won't tell us to highlight, so we'll have to highlight ourselves}
001355              IF SELF.view.OKToDrawIn(SELF.textImage.extentLRect) THEN
001356                  SELF.panel.Highlight(SELF, hOffToOn);
001357              IF SELF.typingCmd <> NIL THEN
001358                  IF SELF.typingCmd.otherInsPts <> NIL THEN
001359                      SELF.typingCmd.otherInsPts.Each(HiliteOtherInsPts);
001360              END;
001361          deferUpdate := FALSE;
001362          SELF.justReturned := FALSE;
001363          {$IFC fTrace}EP;{$ENDC}
001364      END;
001365
001366
001367  {$S SgTxtWrm}
001368      PROCEDURE TInsertionPoint.KeyReturn;
001369      VAR para1:      TEditPara;
001370          newPara:    TEditPara;
001371          selSize:    INTEGER;
001372
001373          PROCEDURE InsPara;
001374          VAR styleChange:    TStyleChange;
001375          BEGIN
001376              {$IFC fTrace}BP(10);{$ENDC}
001377              para1 := SELF.textRange.firstPara;
001378              selSize := para1.size - SELF.textRange.firstLP;
001379              newPara := SELF.textImage.NewEditPara(selSize, para1.format);
001380              newPara.StartEdit(newPara.GrowSize);
001381              newPara.ReplPara(0, 0, para1, SELF.textRange.firstLP, selSize);
001382              IF TFakeTStyle(newPara.format.dfltTStyle) <> TFakeTStyle(SELF.currTypeStyle) THEN
001383                  BEGIN
001384                  styleChange.lp := -1;
001385                  styleChange.newStyle := SELF.currTypeStyle;
001386                  newPara.typeStyles.PutAt(1, @styleChange);
001387                  END;
```

```
001388              para1.ReplPString(SELF.textRange.firstLP, selSize, NIL);
001389              para1.StopEdit;
001390              SELF.textImage.text.InsParaAfter(para1, newPara);
001391              {$IFC fTrace}EP;{$ENDC}
001392          END;
001393
001394          PROCEDURE Adjust;
001395              PROCEDURE AdjustOtherInsPts(obj: TObject);
001396              VAR insertPt:   TInsertionPoint;
001397              BEGIN
001398                  insertPt := TInsertionPoint(obj);
001399                  WITH insertPt, textRange DO
001400                      BEGIN
001401                      firstLP := 0;
001402                      lastLP := 0;
001403                      firstIndex := SELF.textRange.firstIndex;
001404                      lastIndex := firstIndex;
001405                      firstPara := newPara;
001406                      lastPara := newPara;
001407                      END;
001408              END;
001409
001410          BEGIN
001411              {$IFC fTrace}BP(10);{$ENDC}
001412              WITH SELF.textRange DO
001413                  BEGIN
001414                  firstPara := newPara;
001415                  lastPara := newPara;
001416                  firstIndex := firstIndex + 1;
001417                  lastIndex := firstIndex;
001418                  firstLP := 0;
001419                  lastLP := 0;
001420                  END;
001421              SELF.newestLP := 0;
001422              SELF.justReturned := TRUE;
001423              WITH SELF.typingCmd DO
001424                  newParaCount := newParaCount + 1;
001425              IF SELF.typingCmd.otherInsPts <> NIL THEN
001426                  SELF.typingCmd.otherInsPts.Each(AdjustOtherInsPts);
001427              deferUpdate := FALSE;
001428              SELF.textImage.useFirstPixel := FALSE;
001429              {$IFC fTrace}EP;{$ENDC}
001430          END;
001431
001432          PROCEDURE HiliteOtherInsPts(obj: TObject);
001433          VAR insertPt:   TInsertionPoint;
001434          BEGIN
001435              insertPt := TInsertionPoint(obj);
```

```
001436              IF insertPt.view.OKToDrawIn(insertPt.textImage.extentLRect) THEN
001437                  insertPt.panel.Highlight(insertPt, hOffToOn);
001438          END;
001439
001440      BEGIN
001441          {$IFC fTrace}BP(10);{$ENDC}
001442          SELF.KeyText;
001443          SELF.ChangeText(InsPara, Adjust);
001444          {If view.OKToDrawIn was TRUE then we won't be going through the update cycle and hence the
001445           ToolKit won't tell us to highlight, so we'll have to highlight ourselves}
001446          IF SELF.view.OKToDrawIn(SELF.textImage.extentLRect) THEN
001447              SELF.panel.Highlight(SELF, hOffToOn);
001448          IF SELF.typingCmd.otherInsPts <> NIL THEN
001449              SELF.typingCmd.otherInsPts.Each(HiliteOtherInsPts);
001450          {$IFC fTrace}EP;{$ENDC}
001451      END;
001452
001453
001454  {$S SgTxtCld}
001455      PROCEDURE TInsertionPoint.KeyTab;
001456      BEGIN
001457          {$IFC fTrace}BP(10);{$ENDC}
001458          SELF.KeyChar(' ');
001459          {$IFC fTrace}EP;{$ENDC}
001460      END;
001461
001462
001463  {$S SgTxtHot}
001464      PROCEDURE TInsertionPoint.MarkChanged;
001465      BEGIN
001466          {$IFC fTrace}BP(10);{$ENDC}
001467          IF SELF.newestLP < SELF.textRange.firstLP THEN
001468              BEGIN
001469              SELF.textRange.firstLP := SELF.newestLP;
001470              SELF.textRange.lastLP := SELF.newestLP;
001471              SUPERSELF.MarkChanged;
001472              END
001473          ELSE
001474              BEGIN
001475              SUPERSELF.MarkChanged;
001476              SELF.textRange.firstLP := SELF.newestLP;
001477              SELF.textRange.lastLP := SELF.newestLP;
001478              END;
001479          {$IFC fTrace}EP;{$ENDC}
001480      END;
001481
001482
001483  {$S SgTxtHot}
```

```
001484      PROCEDURE TInsertionPoint.MouseMove(mouseLPt: LPoint);
001485      VAR currPImage:         TParaImage;
001486          currLP:             INTEGER;
001487          multiParaSelection: TMultiParaSelection;
001488          oneParaSelection:   TOneParaSelection;
001489          textImage:          TTextImage;
001490          firstTxtImg:        TTextImage;
001491          paraIndex:          LONGINT;
001492          wasParaSel:         BOOLEAN;
001493
001494      BEGIN
001495          {$IFC fTrace}BP(10);{$ENDC}
001496          SELF.currLPt := mouseLPt;
001497          IF NOT EqualLPt(mouseLPt, SELF.anchorLPt) THEN
001498              BEGIN
001499              textImage := SELF.textImage.FindTextImage(mouseLPt, firstTxtImg);
001500              textImage.FindParaAndLp(mouseLPt, currPImage, paraIndex, currLP);
001501
001502              IF (currPImage.paragraph <> SELF.textRange.firstPara) THEN
001503                  BEGIN
001504                  {Turn insertion point off if necessary}
001505                  wasParaSel := SELF.isParaSelection;
001506                  IF NOT wasParaSel AND (paraIndex > SELF.textRange.firstIndex) THEN
001507                      SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, FALSE);
001508                  {call new with same paragraph, followed by MouseMove so highlighting will work correctly
001509                   and so MouseMove can figure out if currPImage is before of after SELF.textRange.firstPara}
001510                  WITH SELF, textRange DO
001511                      {$H-} {ought to be safe: no VAR params, etc}
001512                      multiParaSelection := TMultiParaSelection(SELF.FreedAndReplacedBy(
001513                                              TMultiParaSelection.CREATE(NIL, SELF.Heap,
001514                                                  view, firstTxtImg, anchorLPt,
001515                                                  firstPara, firstIndex, firstLP,
001516                                                  firstPara, firstIndex, firstLP, TRUE)));
001517                      {$H+}
001518                  multiParaSelection.isParaSelection := wasParaSel;
001519                  multiParaSelection.MouseMove(mouseLPt);
001520                  END
001521              ELSE IF (currLP <> SELF.textRange.firstLP) THEN
001522                  BEGIN
001523                  {Turn insertion point off first}
001524                  SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, FALSE);
001525                  {call CREATE with same LP for begin and end,
001526                      followed by MouseMove so highlighting will work correctly}
001527                  WITH SELF, textRange DO
001528                      {$H-} {ought to be safe: no VAR params, etc}
001529                      oneParaSelection := TOneParaSelection(SELF.FreedAndReplacedBy(
001530                                              TOneParaSelection.CREATE(NIL, SELF.Heap,
001531                                                  view, firstTxtImg, anchorLPt,
```

```
001532                                              firstPara, firstIndex, firstLP, firstLP)));
001533                  {$H+}
001534              oneParaSelection.MouseMove(mouseLPt);
001535              END;
001536          END;
001537      {$IFC fTrace}EP;{$ENDC}
001538  END;
001539
001540
001541  {$S SgTxtWrm}
001542      PROCEDURE TInsertionPoint.MousePress(mouseLPt: LPoint);
001543      VAR first, last:        INTEGER;
001544          oneParaSelection:   TOneParaSelection;
001545          funnnyInsPoint:     TInsertionPoint;
001546      BEGIN
001547          {$IFC fTrace}BP(10);{$ENDC}
001548          IF clickState.fShift THEN
001549              SELF.MouseMove(mouseLPt)
001550          ELSE IF clickState.clickCount = 2 THEN  {double click}
001551              BEGIN
001552              {This check should solve the problem of double clicking on the last half of the last
001553               character of a word and not getting the word selected}
001554              WITH SELF.textRange DO
001555                  BEGIN
001556                  first := firstLP;
001557                  {$H-}
001558                  IF NOT firstPara.Qualifies(first) THEN
001559                      IF first > 0 THEN
001560                          first := first - 1;
001561                  {$H+}
001562                  END;
001563              SELF.textRange.firstPara.FindWordBounds(first, first, last);
001564              IF first <> last THEN
001565                  BEGIN
001566                  {Turn insertion point off first}
001567                  SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, FALSE);
001568                  WITH SELF, textRange DO
001569                      {$H-}
001570                      oneParaSelection := TOneParaSelection(SELF.FreedAndReplacedBy(
001571                                              TOneParaSelection.CREATE(NIL, SELF.Heap,
001572                                                  view, textImage, anchorLPt,
001573                                                  firstPara, firstIndex, first, last)));
001574                      {$H+}
001575                  WITH oneParaSelection DO
001576                      BEGIN
001577                      anchorbegin := first;
001578                      anchorEnd := last;
001579                      isWordSelection := TRUE;
```

```
001580                  END;
001581              oneParaSelection.textImage.text.HiliteRange(hOffToOn, oneParaSelection.textRange, FALSE);
001582              END;
001583          END
001584      ELSE IF clickState.clickCount = 3 THEN  {triple click, happens if double click didn't select word}
001585          BEGIN
001586          {Turn insertion point off first}
001587          SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, SELF.isParaSelection);
001588
001589          IF SELF.textRange.firstPara.size = 0 THEN
001590          {This case precipitates the odd notion of an "insertion point" the width of the textImage;
001591           (ie. when one triple clicks on an empty paragraph)  A special check in IdleBegin makes sure
001592           it doesn't blink.}
001593              BEGIN
001594              WITH SELF, textRange DO
001595                  {$H-}
001596                  funnnyInsPoint := TInsertionPoint(SELF.FreedAndReplacedBy(
001597                                          TInsertionPoint.CREATE(NIL, SELF.Heap,
001598                                              view, textImage, anchorLPt,
001599                                              firstPara, firstIndex, 0)));
001600                  {$H+}
001601              WITH funnnyInsPoint DO
001602                  BEGIN
001603                  isParaSelection := TRUE;
001604                  isWordSelection := FALSE;
001605                  END;
001606              funnnyInsPoint.textImage.text.HiliteRange(hOffToOn, SELF.textRange, TRUE);
001607              END
001608          ELSE
001609              BEGIN
001610              WITH SELF, textRange DO
001611                  {$H-}
001612                  oneParaSelection := TOneParaSelection(SELF.FreedAndReplacedBy(
001613                                          TOneParaSelection.CREATE(NIL, SELF.Heap,
001614                                              view, textImage, anchorLPt,
001615                                              firstPara, firstIndex, 0, firstPara.size)));
001616                  {$H+}
001617              WITH oneParaSelection DO
001618                  BEGIN
001619                  isParaSelection := TRUE;
001620                  isWordSelection := FALSE;
001621                  anchorBegin := 0;
001622                  anchorEnd := textRange.lastLP;
001623                  END;
001624              oneParaSelection.textImage.text.HiliteRange(hOffToOn, oneParaSelection.textRange, TRUE);
001625              END;
001626          END
001627      ELSE
```

```
001628              SELF.textImage.MousePress(mouseLPt);
001629          {$IFC fTrace}EP;{$ENDC}
001630      END;
001631
001632
001633  {$S SgTxtCld}
001634      FUNCTION TInsertionPoint.NewStyleCmd(heap: THeap; cmdNumber: TCmdNumber;
001635                                           textImage: TTextImage): TCommand;
001636      BEGIN
001637          {$IFC fTrace}BP(10);{$ENDC}
001638          {Do stuff but set NewStyleCmd NIL so last cmd not committed};
001639          SELF.styleCmdNumber := cmdNumber;
001640          WITH SELF.currTypeStyle, font DO
001641              CASE cmdNumber OF
001642                  uPlain:
001643                      onFaces := [];
001644                  uBold:
001645                      onFaces := onFaces + [bold];
001646                  uItalic:
001647                      onFaces := onFaces + [italic];
001648                  uUnderline:
001649                      onFaces := onFaces + [underlined];
001650                  uShadow:
001651                      onFaces := onFaces + [shadow];
001652                  uOutline:
001653                      onFaces := onFaces + [outline];
001654                  uModern, uClassic:
001655                      font.fontFamily := cmdNumber - uModern + 1;
001656
001657                  {$IFC LibraryVersion <= 20}
001658                  uFnt0, uFnt1, uFnt2, uFnt3, uFnt4, uFnt5, uFnt6, uFnt7, uFnt8, uFnt9, uFnt10, uFnt11:
001659                      CASE cmdNumber OF
001660                          uFnt0:   fontFamily := famSystem;
001661                          uFnt1:
001662                              BEGIN
001663                              fontFamily := famModern;
001664                              fontSize := size15Pitch;
001665                              END;
001666                          uFnt2:
001667                              BEGIN
001668                              fontFamily := famModern;
001669                              fontSize := size12Pitch;
001670                              END;
001671                          uFnt3:
001672                              BEGIN
001673                              fontFamily := famClassic;
001674                              fontSize := size12Pitch;
001675                              END;
```

```
001676                          uFnt4:
001677                              BEGIN
001678                              fontFamily := famModern;
001679                              fontSize := size10Pitch;
001680                              END;
001681                          uFnt5:
001682                              BEGIN
001683                              fontFamily := famModern;
001684                              fontSize := size14Point;
001685                              END;
001686                          uFnt6:
001687                              BEGIN
001688                              fontFamily := famModern;
001689                              fontSize := size12Point;
001690                              END;
001691                          uFnt7:
001692                              BEGIN
001693                              fontFamily := famClassic;
001694                              fontSize := size12Point;
001695                              END;
001696                          uFnt8:
001697                              BEGIN
001698                              fontFamily := famModern;
001699                              fontSize := size18Point;
001700                              END;
001701                          uFnt9:
001702                              BEGIN
001703                              fontFamily := famClassic;
001704                              fontSize := size18Point;
001705                              END;
001706                          uFnt10:
001707                              BEGIN
001708                              fontFamily := famModern;
001709                              fontSize := size24Point;
001710                              END;
001711                          uFnt11:
001712                              BEGIN
001713                              fontFamily := famClassic;
001714                              fontSize := size24Point;
001715                              END;
001716                          END;
001717                  {$ENDC}
001718
001719                  OTHERWISE
001720                      font.fontSize := cmdNumber - u20Pitch + 1;
001721                  END;
001722          NewStyleCmd := NIL;
001723          {$IFC fTrace}EP;{$ENDC}
```

```
001724      END;
001725
001726
001727
001728  {$S SgTxtCld}
001729      FUNCTION TInsertionPoint.NewCutCopyCmd(heap: THeap; cmdNumber: TCmdNumber;
001730                                          textImage: TTextImage): TCommand;
001731      BEGIN
001732          {$IFC fTrace}BP(10);{$ENDC}
001733          {don't create a new command object for insertion point style change}
001734          NewCutCopyCmd := NIL;
001735          {$IFC fTrace}EP;{$ENDC}
001736      END;
001737
001738
001739  {$S SgTxtCld}
001740      FUNCTION TInsertionPoint.SelSize: INTEGER;
001741      BEGIN
001742          {$IFC fTrace}BP(9);{$ENDC}
001743          SelSize := 0;
001744          {$IFC fTrace}EP;{$ENDC}
001745      END;
001746
001747
001748  {$S SgTxtHot}
001749      PROCEDURE TInsertionPoint.StyleFromContext;
001750      VAR typeStyle:      TTypeSTyle;
001751      BEGIN
001752          {$IFC fTrace}BP(9);{$ENDC}
001753          SELF.textRange.firstPara.StyleAt(Max(SELF.textRange.firstLP - 1, 0), typeStyle);
001754          SELF.currTypeStyle := typeStyle;
001755          {$IFC fTrace}EP;{$ENDC}
001756      END;
001757
001758
001759  {$S SgTxtIni}
001760  BEGIN
001761      {temp patch to get fonts from universal text}
001762      uvFont[4].fontFamily := famModern;
001763      uvFont[5].fontFamily := famModern;
001764      uvFont[6].fontFamily := famModern;
001765      uvFont[7].fontFamily := famModern;
001766      uvFont[8].fontFamily := famModern;
001767      uvFont[9].fontFamily := famModern;
001768      uvFont[10].fontFamily := famClassic;
001769      uvFont[11].fontFamily := famClassic;
001770      uvFont[12].fontFamily := famClassic;
001771      uvFont[13].fontFamily := famClassic;
```

```
001772      uvFont[14].fontFamily := famClassic;
001773      uvFont[4].fontSize := 5;
001774      uvFont[5].fontSize := 7;
001775      uvFont[6].fontSize := 8;
001776      uvFont[7].fontSize := 2;
001777      uvFont[8].fontSize := 3;
001778      uvFont[9].fontSize := 4;
001779      uvFont[10].fontSize := 5;
001780      uvFont[11].fontSize := 7;
001781      uvFont[12].fontSize := 8;
001782      uvFont[13].fontSize := 3;
001783      uvFont[14].fontSize := 6;
001784  END; {Methods of TInsertionPoint}
001785
001786
001787  METHODS OF TOneParaSelection;
001788
001789  {$S SgTxtHot}
001790      FUNCTION  TOneParaSelection.CREATE(object: TObject; heap: THeap; itsView: TView; itsTextImage: TTextImage;
001791                                         itsAnchorLPt: LPoint; itsParagraph: TEditPara; itsIndex: LONGINT;
001792                                         oldLP: INTEGER; currLP: INTEGER): TOneParaSelection;
001793      BEGIN
001794          {$IFC fTrace}BP(10);{$ENDC}
001795          IF object = NIL THEN
001796              object := NewObject(heap, THISCLASS);
001797          SELF := TOneParaSelection(TTextSelection.CREATE(object, heap, itsView, itsTextImage, itsAnchorLPt,
001798                                                itsParagraph, itsIndex, Min(oldLP, currLP),
001799                                                itsParagraph, itsIndex, Max(oldLP, currLP)));
001800
001801          WITH SELF DO
001802              BEGIN
001803              anchorBegin := oldLP;
001804              anchorEnd := oldLP;
001805              viewTick := -1; { force recalculation of extent }
001806              END;
001807          {$IFC fTrace}EP;{$ENDC}
001808      END;
001809
001810
001811  {$S SgTxtCld}
001812  {$IFC fTextTrace}
001813      PROCEDURE TOneParaSelection.Fields(PROCEDURE Field(nameAndType: S255));
001814      BEGIN
001815          TTextSelection.Fields(Field);
001816          Field('anchorBegin: INTEGER');
001817          Field('anchorEnd: INTEGER');
001818          Field('');
001819      END;
```

```
001820  {$ENDC}
001821
001822  {$S SgTxtCld}
001823      PROCEDURE TOneParaSelection.ChangeStyle(cmdNumber: TCmdNumber);
001824      VAR newTypeStyle:   TTypeStyle;
001825      BEGIN
001826          {$IFC fTrace}BP(10);{$ENDC}
001827          WITH SELF.textRange DO
001828              {$H-}
001829              SELF.DoChangeStyle(cmdNumber, firstPara, firstLP, lastLP, newTypeStyle);
001830              {$H+}
001831          SELF.currTypeStyle := newTypeStyle;
001832          {$IFC fTrace}EP;{$ENDC}
001833      END;
001834
001835
001836  {$S SgTxtCld}
001837      {CopySelf is used for copying to the clipboard}
001838      FUNCTION TOneParaSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection;
001839      VAR paragraph:      TEditPara;
001840          selSize:        INTEGER;
001841          newImage:       TParaImage;
001842          lastLine:       TLineInfo;
001843          textImage:      TTextImage;
001844          text:           TText;
001845          imageLRect:     LRect;
001846      BEGIN
001847          {$IFC fTrace}BP(11);{$ENDC}
001848          text := TText.CREATE(NIL, heap, TStyleSheet.CREATE(NIL, heap));
001849          imageLRect := view.extentLRect;
001850          InsetLRect(imageLRect, cHorizMargin, cVertMargin);
001851          textImage := SELF.textImage.TxtImgForClipBoard(heap, view, imageLRect, text, TRUE);
001852          textImage.minHeight := 0;
001853          text.txtImgList.InsLast(textImage);
001854
001855          selSize := SELF.textRange.lastLP-SELF.textRange.firstLP;
001856          paragraph := textImage.NewEditPara(selSize, TParaFormat(SELF.textRange.firstPara.format.Clone(heap)));
001857          paragraph.ReplPara(0, 0, SELF.textRange.firstPara, SELF.textRange.firstLP, selSize);
001858
001859          newImage := textImage.NewParaImage(paragraph, textImage.extentLRect, 0, 0);
001860          textImage.imageList.InsLast(newImage);
001861          textImage.text.paragraphs.InsLast(paragraph);
001862
001863
001864          { make view extentLRect exactly fit the lines }
001865          textImage.RecomputeImages(actionNone, TRUE);
001866          WITH textImage.extentLRect DO
001867              view.extentLRect.bottom := bottom - top + 2 * cVertMargin;
```

```
001868
001869          CopySelf := TMultiParaSelection.CREATE(NIL, heap, view, textImage, zeroLPt, paragraph, 1, 0,
001870                                                 paragraph, 1, selSize, TRUE);
001871          {$IFC fTrace}EP;{$ENDC}
001872      END;
001873
001874
001875  {$S SgTxtWrm}
001876      PROCEDURE TOneParaSelection.DeleteAndFree;
001877      VAR delta:  INTEGER;
001878
001879          PROCEDURE DelText;
001880          BEGIN
001881              IF SELF.isParaSelection AND
001882                          (SELF.textRange.firstIndex <> SELF.textImage.text.paragraphs.size) THEN
001883                  BEGIN
001884                  SELF.textImage.text.DelPara(SELF.textRange.firstPara, FALSE);
001885                  SELF.textImage.text.paragraphs.DelObject(SELF.textRange.firstPara, TRUE);
001886                  END
001887              ELSE
001888                  SELF.textRange.firstPara.ReplPString(SELF.textRange.firstLP,
001889                                  SELF.textRange.lastLP-SELF.textRange.firstLP, NIL);
001890          END;
001891
001892          PROCEDURE Adjust;
001893              PROCEDURE Subtract(paraImage: TParaImage);
001894              BEGIN
001895                  paraImage.AdjustLineLPs(SELF.textRange.firstLP, delta);
001896              END;
001897          BEGIN
001898              IF NOT SELF.isParaSelection THEN
001899                  BEGIN
001900                  delta := SELF.textRange.firstLP - SELF.textRange.lastLP;
001901                  SELF.textRange.firstPara.EachImage(Subtract);
001902                  END;
001903              WITH SELF.textRange DO
001904                  BEGIN
001905                  IF SELF.isParaSelection THEN
001906                      BEGIN
001907                      {$H-}
001908                      firstPara := TEditPara(SELF.textImage.text.paragraphs.At(firstIndex));
001909                      {$H+}
001910                      firstLP := 0;
001911                      lastPara := firstPara;
001912                      END;
001913                  lastLP := firstLP;
001914                  END;
001915          END;
```

```
001916      BEGIN
001917          {$IFC fTrace}BP(10);{$ENDC}
001918          SELF.ChangeText(DelText, Adjust);
001919          {$IFC fTrace}EP;{$ENDC}
001920      END;
001921
001922
001923  {$S SgTxtHot}
001924      FUNCTION  TOneParaSelection.DeleteButSave: TText;
001925      VAR oldPara:    TEditPara;
001926          selSize:    INTEGER;
001927          newPara:    TEditPara;
001928          text:       TText;
001929          PROCEDURE DelText;
001930          BEGIN
001931              oldPara := SELF.textRange.firstPara;
001932              IF SELF.isParaSelection AND
001933                          (SELF.textRange.firstIndex <> SELF.textImage.text.paragraphs.size) THEN
001934                  BEGIN
001935                  SELF.textImage.text.DelPara(oldPara, FALSE);
001936                  SELF.textImage.text.paragraphs.DelObject(oldPara, FALSE);
001937                  newPara := oldPara;
001938                  END
001939              ELSE
001940                  BEGIN
001941                  selSize := SELF.textRange.lastLP-SELF.textRange.firstLP;
001942                  newPara := SELF.textImage.NewEditPara(selSize, oldPara.format);
001943                  newPara.ReplPara(0, 0, oldPara, SELF.textRange.firstLP, selSize);
001944                  oldPara.ReplPString(SELF.textRange.firstLP, selSize, NIL);
001945                  END;
001946              text := TText.CREATE(NIL, SELF.Heap, NIL);
001947              text.paragraphs.InsLast(newpara);
001948          END;
001949
001950          PROCEDURE Adjust;
001951              PROCEDURE Subtract(paraImage: TParaImage);
001952              BEGIN
001953                  paraImage.AdjustLineLPs(SELF.textRange.firstLP, -selSize);
001954              END;
001955          BEGIN
001956              WITH SELF.textRange DO
001957                  BEGIN
001958                  IF SELF.isParaSelection THEN
001959                      BEGIN
001960                      {$H-}
001961                      firstPara := TEditPara(SELF.textImage.text.paragraphs.At(firstIndex));
001962                      {$H+}
001963                      firstLP := 0;
```

```
001964                      lastPara := firstPara;
001965                      END;
001966                  lastLP := firstLP;
001967                  END;
001968              IF NOT SELF.isParaSelection THEN
001969                  SELF.textRange.firstPara.EachImage(Subtract);
001970          END;
001971      BEGIN
001972          {$IFC fTrace}BP(10);{$ENDC}
001973          SELF.ChangeText(DelText, Adjust);
001974          DeleteButSave := text;
001975          {$IFC fTrace}EP;{$ENDC}
001976      END;
001977
001978
001979  {$S SgTxtHot}
001980      PROCEDURE TOneParaSelection.MouseMove(mouseLPt: LPoint);
001981          { assumes highlighting is ON }
001982      VAR currPImage:         TParaImage;
001983          currLP:             INTEGER;
001984          oldLP:              INTEGER;    { end (ie. not the anchor) of indication }
001985          multiParaSelection: TMultiParaSelection;
001986          textImage:          TTextImage;
001987          firstTxtImg:        TTextImage;
001988          first, last:        INTEGER;
001989          paraSelect:         BOOLEAN;
001990          paraIndex:          LONGINT;
001991
001992          PROCEDURE HiExtOnPads(highTransit: THighTransit; startLP,endLP: INTEGER);
001993          BEGIN
001994              SELF.textImage.text.HiliteParagraphs(highTransit, SELF.textRange.firstIndex, startLP,
001995                                                                SELF.textRange.firstIndex, endLP,
001996                                                                SELF.isParaSelection);
001997          END;
001998      BEGIN
001999          {$IFC fTrace}BP(10);{$ENDC}
002000          SELF.currLPt := mouseLPt;
002001
002002          textImage := SELF.textImage.FindTextImage(mouseLPt, firstTxtImg);
002003          textImage.FindParaAndLp(mouseLPt, currPImage, paraIndex, currLP);
002004
002005          IF currPImage.paragraph <> SELF.textRange.firstPara THEN
002006              BEGIN
002007              {call new with same paraImage, followed by MouseMove so highlighting will work correctly}
002008              paraSelect := SELF.isParaSelection;
002009              first := SELF.anchorBegin;
002010              last := SELF.anchorEnd;
002011              WITH SELF, textRange DO
```

```
002012                {$H-}
002013                multiParaSelection := TMultiParaSelection(SELF.FreedAndReplacedBy(
002014                                      TMultiParaSelection.CREATE(NIL, SELF.Heap,
002015                                          view, firstTxtImg, anchorLPt,
002016                                          firstPara, firstIndex, firstLP,
002017                                          firstPara, firstIndex, lastLP, firstLP = anchorBegin)));
002018                {$H+}
002019            WITH multiParaSelection DO
002020                BEGIN
002021                isParaSelection := paraSelect;
002022                anchorBegin := first;
002023                anchorEnd := last;
002024                isWordSelection := NOT paraSelect AND (first <> last);
002025                END;
002026            multiParaSelection.MouseMove(mouseLPt);
002027            END
002028        ELSE IF NOT SELF.isParaSelection THEN
002029            BEGIN
002030            IF SELF.isWordSelection THEN
002031                BEGIN
002032                {So dragging over last half of last character of word doesn't select space}
002033                WITH SELF.textRange DO
002034                    BEGIN
002035                    {$H-}
002036                    IF NOT firstPara.Qualifies(currLP) THEN
002037                        IF currLP > 0 THEN
002038                            currLP := currLP - 1;
002039                    {$H+}
002040                    END;
002041                SELF.textRange.firstPara.FindWordBounds(currLP, first, last);
002042                IF first <= SELF.anchorBegin THEN
002043                    currLP := first
002044                ELSE
002045                    currLP := last;
002046                END;
002047
002048            IF currLP <= SELF.anchorBegin THEN
002049                BEGIN
002050                oldLP := SELF.textRange.firstLP;
002051                IF SELF.anchorEnd <> SELF.textRange.lastLP THEN
002052                    HiExtOnPads(hOnToOff, SELF.anchorEnd, SELF.textRange.lastLP);
002053                SELF.textRange.firstLP := currLP;
002054                SELF.textRange.lastLP := SELF.anchorEnd;
002055                END
002056            ELSE
002057                BEGIN
002058                oldLP := SELF.textRange.lastLP;
002059                IF SELF.anchorBegin <> SELF.textRange.firstLP THEN
```

```
002060                  HiExtOnPads(hOnToOff, SELF.textRange.firstLP, SELF.anchorBegin);
002061              SELF.textRange.firstLP := SELF.anchorBegin;
002062              SELF.textRange.lastLP := currLP;
002063              END;
002064
002065          IF currLP <> oldLP THEN
002066              IF currLP < oldLP THEN
002067                  HiExtOnPads(hOffToOn, currLP, oldLP)
002068              ELSE
002069                  HiExtOnPads(hOffToOn, oldLP, currLP);
002070
002071          END;
002072      {$IFC fTrace}EP;{$ENDC}
002073  END;
002074
002075
002076  {$S SgTxtHot}
002077      PROCEDURE TOneParaSelection.MousePress(mouseLPt: LPoint);
002078      VAR first, last:        INTEGER;
002079          oneParaSelection:   TOneParaSelection;
002080          multiParaSelection: TMultiParaSelection;
002081      BEGIN
002082          {$IFC fTrace}BP(10);{$ENDC}
002083          IF clickState.fShift THEN
002084              SELF.MouseMove(mouseLPt)
002085          ELSE IF clickState.clickCount = 2 THEN  {double click}
002086              BEGIN
002087              {should select words at beginning and end of current selection (later)}
002088              SELF.textRange.firstPara.FindWordBounds(SELF.anchorBegin, first, last);
002089              SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, SELF.isParaSelection);
002090              WITH SELF, textRange DO
002091                  {$H-}
002092                  oneParaSelection := TOneParaSelection(SELF.FreedAndReplacedBy(
002093                                          TOneParaSelection.CREATE(NIL, SELF.Heap,
002094                                              view, textImage, anchorLPt,
002095                                              firstPara, firstIndex, first, last)));
002096                  {$H+}
002097              WITH oneParaSelection DO
002098                  BEGIN
002099                  anchorBegin := first;
002100                  anchorEnd := last;
002101                  isWordSelection := TRUE;
002102                  END;
002103              oneParaSelection.textImage.text.HiliteRange(hOnToOff, oneParaSelection.textRange,
002104                                                                          SELF.isParaSelection);
002105              END
002106          ELSE IF clickState.clickCount = 3 THEN  {triple click}
002107              BEGIN
```

```
002108              {Turn current highlighting off first}
002109              SELF.textImage.text.HiliteRange(hOnToOff, SELF.textRange, SELF.isParaSelection);
002110              WITH SELF, textRange DO
002111                  {$H-}
002112                  oneParaSelection := TOneParaSelection(SELF.FreedAndReplacedBy(
002113                                      TOneParaSelection.CREATE(NIL, SELF.Heap,
002114                                          view, textImage, anchorLPt,
002115                                          firstPara, firstIndex, 0, firstPara.size)));
002116                  {$H+}
002117              WITH oneParaSelection DO
002118                  BEGIN
002119                  isParaSelection := TRUE;
002120                  isWordSelection := FALSE;
002121                  anchorBegin := 0;
002122                  anchorEnd := textRange.lastLP;
002123                  END;
002124              oneParaSelection.textImage.text.HiliteRange(hOnToOff, oneParaSelection.textRange, TRUE);
002125              END
002126          ELSE
002127              SELF.textImage.MousePress(mouseLPt);
002128          {$IFC fTrace}EP;{$ENDC}
002129      END;
002130
002131
002132  {$S SgTxtHot}
002133      PROCEDURE TOneParaSelection.MouseRelease;
002134      VAR insPt:  TInsertionPoint;
002135      BEGIN
002136          {$IFC fTrace}BP(10);{$ENDC}
002137          IF SELF.textRange.firstLP = SELF.textRange.lastLP THEN
002138              BEGIN
002139              insPt := SELF.BecomeInsertionPoint;
002140              insPt.textImage.text.HiliteRange(hOffToOn, insPt.textRange, SELF.isParaSelection);
002141              END
002142          ELSE
002143              SELF.textImage.text.ChangeSelInOtherPanels(SELF);
002144          SUPERSELF.MouseRelease;
002145          {$IFC fTrace}EP;{$ENDC}
002146      END;
002147
002148
002149  {$S SgTxtCld}
002150      FUNCTION TOneParaSelection.SelSize: INTEGER;
002151      BEGIN
002152          {$IFC fTrace}BP(9);{$ENDC}
002153          SelSize := SELF.textRange.lastLP - SELF.textRange.firstLP;
002154          {$IFC fTrace}EP;{$ENDC}
002155      END;
```

```
002156
002157
002158  {$S SgTxtHot}
002159      PROCEDURE TOneParaSelection.StyleFromContext;
002160      VAR typeStyle:  TTypeStyle;
002161      BEGIN
002162          {$IFC fTrace}BP(8);{$ENDC}
002163          SELF.textRange.firstPara.StyleAt(SELF.textRange.firstLP, typeStyle);
002164          SELF.currTypeStyle := typeStyle;
002165          {$IFC fTrace}EP;{$ENDC}
002166      END;
002167
002168
002169  {$S SgTxtIni}
002170  END; {Methods of TOneParaSelection}
002171
002172
002173  METHODS OF TMultiParaSelection;
002174
002175  {$S SgTxtCld}
002176      FUNCTION TMultiParaSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
002177                                          itsTextImage: TTextImage; itsAnchorLPt: LPoint;
002178                                          beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
002179                                          endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER;
002180                                          beginIsAnchor: BOOLEAN): TMultiParaSelection;
002181      BEGIN
002182          {$IFC fTrace}BP(10);{$ENDC}
002183          IF object = NIL THEN
002184              object := NewObject(heap, THISCLASS);
002185          SELF := TMultiParaSelection(TTextSelection.CREATE(object, heap, itsView, itsTextImage, itsAnchorLPt,
002186                                              beginPara, beginIndex, beginLP,
002187                                              endPara, endIndex, endLP));
002188
002189
002190          WITH SELF DO
002191              BEGIN
002192              IF beginIsAnchor THEN
002193                  BEGIN
002194                  anchorPara := beginPara;
002195                  anchorIndex := beginIndex;
002196                  anchorBegin := beginLP;
002197                  anchorEnd := beginLP;
002198                  END
002199              ELSE
002200                  BEGIN
002201                  anchorPara := endPara;
002202                  anchorIndex := endIndex;
002203                  anchorBegin := endLP;
```

```
002204                  anchorEnd := endLP;
002205                  END;
002206              END;
002207          {$IFC fTrace}EP;{$ENDC}
002208      END;
002209
002210  {$S SgTxtCld}
002211  {$IFC fTextTrace}
002212      PROCEDURE TMultiParaSelection.Fields(PROCEDURE Field(nameAndType: S255));
002213      BEGIN
002214          TTextSelection.Fields(Field);
002215          Field('anchorPara: TEditPara');
002216          Field('anchorIndex: LONGINT');
002217          Field('anchorBegin: INTEGER');
002218          Field('anchorEnd: INTEGER');
002219          Field('');
002220      END;
002221  {$ENDC}
002222
002223
002224  {$S SgTxtCld}
002225      PROCEDURE TMultiParaSelection.ChangeStyle(cmdNumber: TCmdNumber);
002226      VAR newTypeStyle:   TTypeStyle;
002227          s:              TListScanner;
002228          paragraph:      TEditPara;
002229          lastPara:       TEditPara;
002230          endRng:         INTEGER;
002231          paraImage:      TParaImage;
002232      BEGIN
002233          {$IFC fTrace}BP(10);{$ENDC}
002234          newTypeStyle := SELF.currTypeStyle;
002235          lastPara := SELF.textRange.lastPara;
002236          s := SELF.textImage.text.paragraphs.ScannerFrom(SELF.textRange.firstIndex-1, scanForward);
002237          WHILE s.Scan(paragraph) DO
002238              IF paragraph = SELF.textRange.firstPara THEN
002239                  BEGIN
002240                  SELF.DoChangeStyle(cmdNumber, paragraph, SELF.textRange.firstLP,
002241                                     paragraph.size, newTypeStyle);
002242                  SELF.currTypeStyle := newTypeStyle;
002243                  IF paragraph = lastPara THEN
002244                      s.Done;
002245                  END
002246              ELSE
002247                  BEGIN
002248                  IF paragraph = lastPara THEN
002249                      BEGIN
002250                      endRng := SELF.textRange.lastLP;
002251                      s.Done;
```

```
002252                  END
002253              ELSE
002254                  endRng := paragraph.size;
002255
002256              SELF.DoChangeStyle(cmdNumber, paragraph, 0, endRng, newTypeStyle);
002257              END;
002258          {$IFC fTrace}EP;{$ENDC}
002259      END;
002260
002261
002262  {$S SgTxtCld}
002263      {CopySelf is used for copying to the clipboard}
002264      FUNCTION TMultiParaSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection;
002265      VAR srcPara:            TEditPara;
002266          cpyPara:            TEditPara;
002267          cpyFirstPara:       TEditPara;
002268          srcLastPara:        TEditPara;
002269          cpyLastPara:        TEditPara;
002270          selSize1:           INTEGER;
002271          selSize2:           INTEGER;
002272          textImage:          TTextImage;
002273          s:                  TListScanner;
002274          text:               TText;
002275          textRange:          TTextRange;
002276          imageLRect:         LRect;
002277      BEGIN
002278          {$IFC fTrace}BP(11);{$ENDC}
002279          text := TText.CREATE(NIL, heap, TStyleSheet.CREATE(NIL, heap));
002280          imageLRect := view.extentLRect;
002281          InsetLRect(imageLRect, cHorizMargin, cVertMargin);
002282          textImage := SELF.textImage.TxtImgForClipBoard(heap, view, imageLRect, text, TRUE);
002283          textImage.minHeight := 0;
002284          text.txtImgList.InsLast(textImage);
002285
002286          textRange := SELF.textRange;
002287          srcPara := textRange.firstPara;
002288          selSize1 := srcPara.size-textRange.firstLP;
002289          cpyPara := textImage.NewEditPara(selSize1, TParaFormat(srcPara.format.clone(heap)));
002290          cpyPara.ReplPara(0, 0, srcPara, textRange.firstLP, selSize1);
002291          cpyFirstPara := cpyPara;
002292
002293          textImage.text.paragraphs.InsLast(cpyFirstPara);
002294
002295          IF textRange.firstPara <> textRange.lastPara THEN
002296              BEGIN
002297              srcLastPara := textRange.lastPara;
002298              selSize2 := textRange.lastLP;
002299              cpyLastPara := textImage.NewEditPara(selSize2, TParaFormat(srcLastPara.format.clone(heap)));
```

```
002300              cpyLastPara.ReplPara(0, 0, srcLastPara, 0, selSize2);
002301
002302              {skip first paragraph by not subtracting one from firstIndex}
002303              s := SELF.textImage.text.paragraphs.ScannerFrom(textRange.firstIndex, scanForward);
002304              WHILE s.Scan(srcPara) DO
002305                  BEGIN
002306                  IF srcPara = textRange.lastPara THEN
002307                      BEGIN
002308                      cpyPara := cpyLastPara;
002309                      s.Done;
002310                      END
002311                  ELSE
002312                      BEGIN
002313                      cpyPara := textImage.NewEditPara(srcPara.size,
002314                                      TParaFormat(srcPara.format.clone(heap)));
002315                      cpyPara.ReplPara(0, 0, srcPara, 0, srcPara.size);
002316                      END;
002317
002318                  textImage.text.paragraphs.InsLast(cpyPara);
002319                  END;
002320              END;
002321
002322          textImage.RecomputeImages(actionNone, TRUE);
002323          WITH textImage.extentLRect DO
002324              view.extentLRect.bottom := bottom - top + 2 * cVertMargin;
002325
002326          CopySelf := TMultiParaSelection.CREATE(NIL, heap, view, textImage, zeroLPt,
002327                                              TEditPara(textImage.text.paragraphs.First), 1, 0,
002328                                              TEditPara(textImage.text.paragraphs.Last),
002329                                              textRange.lastIndex - textRange.firstIndex + 1,
002330                                              selSize2, TRUE);
002331          {$IFC fTrace}EP;{$ENDC}
002332      END;
002333
002334
002335  {$S SgTxtCld}
002336      FUNCTION TMultiParaSelection.Delete(saveIt: BOOLEAN): TText;
002337      VAR firstPara:          TEditPara;
002338          lastPara:           TEditPara;
002339          paragraph:          TEditPara;
002340          text:               TText;
002341          paraList:           TList;
002342          s:                  TListScanner;
002343          selSize:            INTEGER;
002344          textRange:          TTextRange;
002345          numParas:           INTEGER;
002346
002347          PROCEDURE DelText;
```

```
002348          BEGIN
002349              {$IFC fTrace}BP(11);{$ENDC}
002350              {If saveIt is TRUE, we want to save the text we're deleting, so create a text object}
002351              IF saveIt THEN
002352                  BEGIN
002353                  text := TText.CREATE(NIL, SELF.Heap, NIL);
002354                  paraList := text.paragraphs;
002355                  END
002356              ELSE
002357                  text := NIL;
002358
002359              textRange := SELF.textRange;
002360              firstPara := textRange.firstPara;
002361              lastPara := textRange.lastPara;
002362              numParas := SELF.textImage.text.paragraphs.size;
002363
002364              {If the last paragraph is selected, treat this like a non-para selection, so that one
002365               empty para will be left at the end after the delete}
002366              IF textRange.lastIndex = numParas THEN
002367                  SELF.isParaSelection := FALSE;
002368
002369              s := SELF.textImage.text.paragraphs.ScannerFrom(textRange.firstIndex-1, scanForward);
002370              WHILE s.Scan(paragraph) DO
002371                  IF paragraph = firstPara THEN
002372                      BEGIN
002373                      IF SELF.isParaSelection THEN
002374                          {If isParaSelection is TRUE then insert it in our save list and
002375                           delete it from the textImage's list}
002376                          BEGIN
002377                          IF saveIt THEN
002378                              paraList.InsLast(firstPara);
002379                          SELF.textImage.text.DelPara(firstPara, FALSE);
002380                          s.Delete(NOT saveIt);
002381                          END
002382                      ELSE
002383                          {If the beginning of the selection is part of a paragraph, then save
002384                          the characters in a new paragraph and delete them in the old}
002385                          BEGIN
002386                          selSize := firstPara.size - textRange.firstLP;
002387                          IF saveIt THEN
002388                              BEGIN
002389                              paragraph := SELF.textImage.NewEditPara(selSize, firstPara.format);
002390                              paragraph.ReplPara(0, 0, firstPara, textRange.firstLP, selSize);
002391                              paraList.InsLast(paragraph);
002392                              END;
002393                          firstPara.ReplPString(textRange.firstLP, selSize, NIL);
002394                          END;
002395                      IF firstPara = lastPara THEN
```

```
002396                      s.Done
002397                  END
002398              ELSE
002399                  BEGIN
002400                  s.Delete(FALSE);
002401                  IF paragraph = lastPara THEN
002402                      BEGIN
002403                      s.Done;
002404                      IF NOT SELF.isParaSelection THEN
002405                          {If the end of the selection is a part of a paragraph, then save the
002406                           selected characters in a new paragraph and append the rest to the
002407                           first paragraph.  Finally, delete and free the last paragraph }
002408                          BEGIN
002409                          selSize := textRange.lastLP;
002410                          IF saveIt THEN
002411                              BEGIN
002412                              paragraph := SELF.textImage.NewEditPara(selSize, lastPara.format);
002413                              paragraph.ReplPara(0, 0, lastPara, 0, selSize);
002414                              END;
002415                          firstPara.ReplPara(firstPara.size, 0, lastPara, selSize,
002416                                                                  lastPara.size-selSize);
002417                          SELF.textImage.text.DelPara(lastPara, TRUE);
002418                          END
002419                      ELSE
002420                          SELF.textImage.text.DelPara(paragraph, NOT saveIt);
002421                      END
002422                  ELSE
002423                      {Delete entire intermediate paragraphs from the textImage}
002424                      SELF.textImage.text.DelPara(paragraph, NOT saveIt);
002425                  IF saveIt THEN
002426                      paraList.InsLast(paragraph);
002427                  END;
002428          {$IFC fTrace}EP;{$ENDC}
002429      END; {DelText}
002430
002431      PROCEDURE SetRange;
002432      BEGIN
002433          WITH SELF, textRange DO
002434              IF isParaSelection THEN
002435                  BEGIN
002436                  {$H-}
002437                  firstPara := TEditPara(textImage.text.paragraphs.At(firstIndex));
002438                  {$H+}
002439                  firstLP := 0;
002440                  END;
002441
002442          WITH SELF.textRange DO
002443              BEGIN
```

```
002444                  lastPara := firstPara;
002445                  lastIndex := firstIndex;
002446                  lastLP := firstLP;
002447                  END;
002448          END;
002449
002450      BEGIN
002451          {$IFC fTrace}BP(11);{$ENDC}
002452          SELF.ChangeText(DelText, SetRange);
002453          Delete := text;
002454          {$IFC fTrace}EP;{$ENDC}
002455      END;
002456
002457
002458  {$S SgTxtCld}
002459      PROCEDURE TMultiParaSelection.DeleteAndFree;
002460      VAR text:   TText;  {dummy var since Delete returns TText; will always be NIL}
002461      BEGIN
002462          {$IFC fTrace}BP(11);{$ENDC}
002463          text := SELF.Delete(FALSE);
002464          {$IFC fTrace}EP;{$ENDC}
002465      END;
002466
002467
002468  {$S SgTxtCld}
002469      FUNCTION TMultiParaSelection.DeleteButSave: TText;
002470      BEGIN
002471          {$IFC fTrace}BP(11);{$ENDC}
002472          DeleteButSave := SELF.Delete(TRUE);
002473          {$IFC fTrace}EP;{$ENDC}
002474      END;
002475
002476
002477  {$S SgTxtCld}
002478      PROCEDURE TMultiParaSelection.MouseMove(mouseLPt: LPoint);
002479          { assumes highlighting is ON }
002480      VAR endPara:            TEditPara;
002481          currPara:           TEditPara;
002482          currPImage:         TParaImage;
002483          oldPImage:          TParaImage;
002484          oldPara:            TEditPara;
002485          paragraph:          TEditPara;
002486          textrange:          TTextRange;
002487          textImage:          TTextImage;
002488          firstTxtImg:        TTextImage;
002489          s:                  TListScanner;
002490          paraImage:          TParaImage;
002491          newText:            TText;
```

```
002492          currLP:             INTEGER;
002493          oldLP:              INTEGER;    { end (ie. not the anchor) of indication }
002494          mouseBeforeAnchor:  BOOLEAN;    { is mouse LPt before the anchor LPt }
002495          oldBeforeCurr:      BOOLEAN;    { is old end LPt before curr LPt }
002496          beginIsAnchor:      BOOLEAN;
002497          currIndex:          LONGINT;
002498          oldIndex:           LONGINT;
002499          startIndex:         LONGINT;
002500          endIndex:           LONGINT;
002501          startLP:            INTEGER;
002502          endLP:              INTEGER;
002503          first, last:        INTEGER;
002504          selChanged:         BOOLEAN;
002505          {$IFC fTextTrace}
002506          str1:                STR255;
002507          str2:                STR255;
002508          {$ENDC}
002509
002510          PROCEDURE NextIndex(oldIndex: LONGINT; oldLP: INTEGER; VAR newIndex: LONGINT; VAR newLP: INTEGER);
002511          BEGIN
002512              IF oldIndex < SELF.textImage.text.paragraphs.size THEN
002513                  BEGIN
002514                  newIndex := oldIndex + 1;
002515                  newLP := 0;
002516                  END
002517              ELSE
002518                  BEGIN
002519                  newIndex := oldIndex;
002520                  newLP := oldLP;
002521                  END;
002522          END;
002523
002524          PROCEDURE PrevIndex(oldIndex: LONGINT; oldLP: INTEGER; VAR newIndex: LONGINT; VAR newLP: INTEGER);
002525          BEGIN
002526              IF oldIndex > 1 THEN
002527                  BEGIN
002528                  newIndex := oldIndex - 1;
002529                  newLP := TEditPara(SELF.textImage.text.paragraphs.At(newIndex)).size;
002530                  END
002531              ELSE
002532                  BEGIN
002533                  newIndex := oldIndex;
002534                  newLP := oldLP;
002535                  END;
002536          END;
002537
002538          PROCEDURE HiExtOnPads(highTransit: THighTransit;
002539                                startIndex: LONGINT; startLP: INTEGER;
```

```
002540                                  endIndex: LONGINT; endLP: INTEGER);
002541          BEGIN
002542              SELF.textImage.text.HiliteParagraphs(highTransit, startIndex, startLP, endIndex, endLP,
002543                                                                                  SELF.isParaSelection);
002544          END;
002545
002546      BEGIN
002547          {$IFC fTrace}BP(10);{$ENDC}
002548          SELF.currLPt := mouseLPt;
002549
002550          {$IFC fTextTrace}
002551          IF fTextTrace THEN
002552              BEGIN
002553              LIntToHex(ORD(SELF.textRange.firstPara), @str1);
002554              LIntToHex(ORD(SELF.textRange.lastPara), @str2);
002555              Writeln;
002556              Writeln('*** MultiPara MouseMove: firstPara, lastPara = (', str1, ',', str2, ')');
002557              Writeln('*** About to call FindParaAndLp');
002558              END;
002559          {$ENDC}
002560
002561          textRange := SELF.textRange;
002562          textImage := SELF.textImage.FindTextImage(mouseLPt, firstTxtImg);
002563          SELF.textImage := firstTxtImg;
002564          textImage.FindParaAndLp(mouseLPt, currPImage, currIndex, currLP);
002565          currPara := currPImage.paragraph;
002566
002567          IF SELF.isParaSelection THEN
002568              BEGIN
002569              IF currIndex < SELF.anchorIndex THEN
002570                  currLP := 0
002571              ELSE
002572                  currLP := currPara.size;
002573              END
002574          ELSE IF SELF.isWordSelection THEN
002575              BEGIN
002576              currPara.FindWordBounds(currLP, first, last);
002577              IF currIndex < SELF.anchorIndex THEN
002578                  currLP := first
002579              ELSE IF currIndex > SELF.anchorIndex THEN
002580                  currLP := last
002581              ELSE IF first <= SELF.anchorBegin THEN
002582                  currLP := first
002583              ELSE
002584                  currLP := last;
002585              END;
002586
002587          IF currIndex = SELF.anchorIndex THEN
```

```
002588              mouseBeforeAnchor := currLP < SELF.anchorBegin
002589          ELSE
002590              mouseBeforeAnchor := currIndex < SELF.anchorIndex;
002591
002592          beginIsAnchor := (textRange.firstIndex = SELF.anchorIndex) AND (textRange.firstLP = SELF.anchorBegin);
002593
002594          {After determining if the mouse is before or after the anchor position, set up variables for
002595           higlighting below and dehilite any text that was on other side of the anchor previous to this
002596           mouse move}
002597          IF mouseBeforeAnchor THEN
002598              BEGIN
002599              oldIndex := textRange.firstIndex;
002600              oldLP := textRange.firstLP;
002601              IF beginIsAnchor THEN
002602                  {current Position is on other side of anchor, so must dehighlight}
002603                  BEGIN
002604                  IF SELF.isParaSelection THEN
002605                      IF SELF.anchorIndex = textRange.lastIndex THEN
002606                          startIndex := -1
002607                      ELSE
002608                          NextIndex(SELF.anchorIndex, SELF.anchorEnd, startIndex, startLP)
002609                  ELSE
002610                      BEGIN
002611                      startIndex := SELF.anchorIndex;
002612                      startLP := SELF.anchorEnd;
002613                      END;
002614                  IF startIndex > 0 THEN
002615                      HiExtOnPads(hOnToOff, startIndex, startLP, textRange.lastIndex, textRange.lastLP);
002616                  END;
002617              WITH SELF, textRange DO
002618                  BEGIN
002619                  firstPara := currPara;
002620                  firstIndex := currIndex;
002621                  lastPara := anchorPara;
002622                  lastIndex := anchorIndex;
002623                  firstLP := currLP;
002624                  lastLP := anchorEnd;
002625                  END;
002626              END
002627          ELSE
002628              BEGIN
002629              oldIndex := textRange.lastIndex;
002630              oldLp := textRange.lastLP;
002631              IF NOT beginIsAnchor THEN
002632                  BEGIN
002633                  {current Position is on other side of anchor, so must dehighlight}
002634                  IF SELF.isParaSelection THEN
002635                      PrevIndex(SELF.anchorIndex, SELF.anchorBegin, startIndex, startLP)
```

```
002636                  ELSE
002637                      BEGIN
002638                      startIndex := SELF.anchorIndex;
002639                      startLP := SELF.anchorBegin;
002640                      END;
002641                  HiExtOnPads(hOnToOff, textRange.firstIndex, textRange.firstLP, startIndex, startLP);
002642                  END;
002643              WITH SELF, textRange DO
002644                  BEGIN
002645                  lastPara := currPara;
002646                  lastIndex := currIndex;
002647                  firstPara := anchorPara;
002648                  firstIndex := anchorIndex;
002649                  firstLP := anchorBegin;
002650                  lastLP := currLP;
002651                  END;
002652              END;
002653
002654          IF mouseBeforeAnchor = beginIsAnchor THEN
002655              oldBeforeCurr := NOT mouseBeforeAnchor
002656          ELSE IF currIndex = oldIndex THEN
002657              oldBeforeCurr := oldLP < currLP
002658          ELSE
002659              oldBeforeCurr := oldIndex < currIndex;
002660
002661          IF oldBeforeCurr THEN
002662              BEGIN
002663              startIndex := oldIndex;
002664              startLP := oldLP;
002665              endIndex := currIndex;
002666              endLP := currLP;
002667              END
002668          ELSE
002669              BEGIN
002670              startIndex := currIndex;
002671              startLP := currLP;
002672              endIndex := oldIndex;
002673              endLP := oldLP;
002674              END;
002675
002676          selChanged := TRUE;
002677          IF SELF.isParaSelection THEN
002678              IF startIndex = endIndex THEN
002679                  selChanged := FALSE
002680              ELSE IF mouseBeforeAnchor THEN
002681                  PrevIndex(endIndex, endLP, endIndex, endLP)
002682              ELSE
002683                  NextIndex(startIndex, startLP, startIndex, startLP)
```

```
002684          ELSE
002685              selChanged := (startIndex <> endIndex) OR (startLP <> endLP);
002686
002687          IF selChanged THEN
002688              HiExtOnPads(hOffToOn, startIndex, startLP, endIndex, endLP);
002689
002690          {$IFC fTrace}EP;{$ENDC}
002691      END;
002692
002693
002694  {$S SgTxtCld}
002695      PROCEDURE TMultiParaSelection.MousePress(mouseLPt: LPoint);
002696      BEGIN
002697          {$IFC fTrace}BP(10);{$ENDC}
002698          IF clickState.fShift THEN
002699              SELF.MouseMove(mouseLPt)
002700          ELSE IF clickState.clickCount > 1 THEN
002701              BEGIN
002702              {For now do nothing if some jerk starts double/triple clicking while dragging}
002703              END
002704          ELSE
002705              SELF.textImage.MousePress(mouseLPt);
002706          {$IFC fTrace}EP;{$ENDC}
002707      END;
002708
002709
002710  {$S SgTxtCld}
002711      PROCEDURE TMultiParaSelection.MouseRelease;
002712      VAR insPt:      TInsertionPoint;
002713          oneParaSel: TOneParaSelection;
002714          first,last: INTEGER;
002715          isPara:     BOOLEAN;
002716      BEGIN
002717          {$IFC fTrace}BP(10);{$ENDC}
002718          IF SELF.textRange.firstPara = SELF.textRange.lastPara THEN
002719              BEGIN
002720              isPara := SELF.isParaSelection;
002721              IF SELF.textRange.firstLP = SELF.textRange.lastLP THEN
002722                  BEGIN
002723                  insPt := SELF.BecomeInsertionPoint;
002724                  insPt.isParaSelection := isPara;
002725                  IF NOT isPara THEN
002726                      insPt.textImage.text.HiliteRange(hOffToOn, insPt.textRange, FALSE);
002727                  END
002728              ELSE
002729                  BEGIN
002730                  first := SELF.anchorBegin;
002731                  last := SELF.anchorEnd;
```

```
002732                  WITH SELF, textRange DO
002733                      {$H-}
002734                      oneParaSel := TOneParaSelection(SELF.FreedAndReplacedBy(TOneParaSelection.CREATE(NIL,
002735                                          SELF.Heap, view, textImage, anchorLPoint,
002736                                          firstPara, firstIndex, firstLP, lastLP)));
002737                      {$H+}
002738                  WITH oneParaSel DO
002739                      BEGIN
002740                      anchorBegin := first;
002741                      anchorEnd := last;
002742                      isParaSelection := isPara;
002743                      isWordSelection := NOT isPara AND (first <> last);
002744                      END;
002745                  SELF.textImage.text.ChangeSelInOtherPanels(oneParaSel);
002746                  END;
002747              END
002748          ELSE
002749              SELF.textImage.text.ChangeSelInOtherPanels(SELF);
002750          SUPERSELF.MouseRelease;
002751          {$IFC fTrace}EP;{$ENDC}
002752      END;
002753
002754
002755  {$S SgTxtCld}
002756      FUNCTION TMultiParaSelection.SelSize: INTEGER;
002757      VAR size:       INTEGER;
002758          s:          TListScanner;
002759          paragraph:  TEditPara;
002760      BEGIN
002761          {$IFC fTrace}BP(9);{$ENDC}
002762          IF SELF.textRange.firstPara = SELF.textRange.lastPara THEN
002763              size := SELF.textRange.lastLP - SELF.textRange.firstLP
002764          ELSE
002765              BEGIN
002766              size := SELF.textRange.firstPara.size - SELF.textRange.firstLP;
002767              {skip first paragraph by not subtracting one from firstIndex}
002768              s := SELF.textImage.text.paragraphs.ScannerFrom(SELF.textRange.firstIndex, scanForward);
002769              WHILE s.Scan(paragraph) DO
002770                  IF paragraph = SELF.textRange.lastPara THEN
002771                      BEGIN
002772                      size := size + SELF.textRange.lastLP;
002773                      s.Done;
002774                      END
002775                  ELSE
002776                      size := size + paragraph.size;
002777              END;
002778          SelSize := size;
002779          {$IFC fTrace}EP;{$ENDC}
```

```
002780      END;
002781
002782
002783  {$S SgTxtCld}
002784      PROCEDURE TMultiParaSelection.StyleFromContext;
002785      VAR typeStyle:  TTypeStyle;
002786      BEGIN
002787          {$IFC fTrace}BP(8);{$ENDC}
002788          SELF.textRange.firstPara.StyleAt(SELF.textRange.firstLP, typeStyle);
002789          SELF.currTypeStyle := typeStyle;
002790          {$IFC fTrace}EP;{$ENDC}
002791      END;
002792
002793
002794  {$S SgTxtIni}
002795  END; {Methods of TMultiParaSelection}
002796
002797
002798  {$S SgTxtCld}
002799
002800  METHODS OF TClearTextCmd;
002801
002802      FUNCTION TClearTextCmd.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
002803                                    itsImage: TImage; itsText: TText): TClearTextCmd;
002804      BEGIN
002805          {$IFC fTrace}BP(11);{$ENDC}
002806          IF object = NIL THEN
002807              object := NewObject(heap, THISCLASS);
002808          SELF := TClearTextCmd(TCommand.CREATE(object, heap, itsCmdNumber, itsImage, TRUE, revealAll));
002809          WITH SELF DO
002810              BEGIN
002811              savedText := NIL;
002812              text := itsText;
002813              END;
002814          {$IFC fTrace}EP;{$ENDC}
002815      END;
002816
002817      PROCEDURE TClearTextCmd.Free;
002818      BEGIN
002819          {$IFC fTrace}BP(10);{$ENDC}
002820          IF SELF.savedText <> NIL THEN
002821              SELF.savedText.FreeSelf(FALSE);
002822          SUPERSELF.Free;
002823          {$IFC fTrace}EP;{$ENDC}
002824      END;
002825
002826  {$IFC fTextTrace}
002827      PROCEDURE TClearTextCmd.Fields(PROCEDURE Field(nameAndType: S255));
```

```
002828      BEGIN
002829          SUPERSELF.Fields(Field);
002830          Field('savedText: TText');
002831          Field('text: TText');
002832          Field('');
002833      END;
002834  {$ENDC}
002835
002836      PROCEDURE TClearTextCmd.Commit;
002837      BEGIN
002838          {$IFC fTrace}BP(10);{$ENDC}
002839          Free(SELF.savedText);
002840          SELF.savedText := NIL;
002841          {$IFC fTrace}EP;{$ENDC}
002842      END;
002843
002844
002845      PROCEDURE TClearTextCmd.Perform(cmdPhase: TCmdPhase);
002846      var textSel:        TTextSelection;
002847          insertionPt:    TInsertionPoint;
002848          text:           TText;
002849          selection:      TSelection;
002850          panel:          TPanel;
002851          {$IFC fTextTrace}
002852          junk:           TObject;
002853          str1:           STR255;
002854          str2:           STR255;
002855          {$ENDC}
002856
002857
002858      BEGIN
002859          {$IFC fTrace}BP(10);{$ENDC}
002860          panel := SELF.image.view.panel;
002861          text := SELF.text;
002862          CASE cmdPhase OF
002863              doPhase, redoPhase:
002864                  BEGIN
002865                  selection := panel.selection;
002866                  WHILE selection.coSelection <> NIL DO
002867                      selection := selection.coSelection;
002868
002869                  {$IFC fTextTrace}
002870                  IF fTextTrace THEN
002871                      BEGIN
002872                      LIntToHex(ORD(selection), @str1);
002873                      Writeln('*** Clear Cmd Perfrom; panel last coselection = ', str1);
002874                      junk := SELF.text.paragraphs.First; {So can set break point here}
002875                      END;
```

```
002876                  {$ENDC}
002877
002878                  textSel := TTextSelection(selection.FreedAndReplacedBy(
002879                                      text.SelectAll(TTextSelection(selection).textImage)));
002880                  text.ChangeSelInOtherPanels(textSel);
002881                  text.HiliteRange(hOffToOn, textSel.textRange, FALSE);
002882                  text := textSel.DeleteButSave;
002883                  SELF.savedText := text;
002884                  insertionPt := textSel.BecomeInsertionPoint;
002885                  {$IFC fTextTrace}
002886                  IF fTextTrace THEN
002887                      BEGIN
002888                      LIntToHex(ORD(insertionPt), @str1);
002889                      Writeln('*** Clear Cmd Perfrom; final insertionPt = ', str1);
002890                      junk := SELF.text.paragraphs.First; {So can set break point here}
002891                      END;
002892                  {$ENDC}
002893
002894                  END;
002895              undoPhase:
002896                  BEGIN
002897                  selection := panel.selection;
002898                  WHILE selection.coSelection <> NIL DO
002899                      selection := selection.coSelection;
002900
002901                  insertionPt := TInsertionPoint(selection);
002902                  insertionPt.InsertText(SELF.savedText, FALSE, FALSE, FALSE);
002903                  Free(SELF.savedText);
002904                  SELF.savedtext := NIL;
002905                  {$$ Need to hilte before, after?}
002906                  SELF.text.ChangeSelInOtherPanels(insertionPt);
002907                  END;
002908              END;
002909          {$IFC fTrace}EP;{$ENDC}
002910      END;
002911
002912  {$S SgTxtIni}
002913  END; {METHODS OF TClearTextCmd}
002914  {$S SgTxtCld}
002915
002916
002917  METHODS OF TStyleCmd;
002918
002919      FUNCTION TStyleCmd.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
002920                                  itsImage: TImage;
002921                                  itsFirstIndex: LONGINT; itsLastIndex:  LONGINT;
002922                                  itsLPFirst: INTEGER; itsLPLast: INTEGER;
002923                                  itsSelection: TTextSelection): TStyleCmd;
```

```
002924      VAR sel:    TTextSelection;
002925      BEGIN
002926          {$IFC fTrace}BP(11);{$ENDC}
002927          IF object = NIL THEN
002928              object := NewObject(heap, THISCLASS);
002929          SELF := TStyleCmd(TCommand.CREATE(object, heap, itsCmdNumber, itsImage, TRUE, revealSome));
002930          sel := TTextSelection(itsSelection.Clone(SELF.Heap));
002931          WITH SELF DO
002932              BEGIN
002933              textSelection := sel;
002934              text := sel.textImage.text;
002935              firstFiltParaIndex := itsFirstIndex;
002936              lastFiltParaIndex := itsLastIndex;
002937              filtFirstLP := itsLPFirst;
002938              filtLastLP := itsLPLast;
002939              currFilteredPara := NIL;
002940              filteredStyles := NIL;
002941              END;
002942          {$IFC fTrace}EP;{$ENDC}
002943      END;
002944
002945      PROCEDURE TStyleCmd.Free;
002946      VAR sPar:       TListScanner;
002947          paragraph:  TEditPara;
002948          paraImage:  TParaImage;
002949
002950      BEGIN
002951          {$IFC fTrace}BP(10);{$ENDC}
002952          sPar := SELF.text.paragraphs.ScannerFrom(SELF.firstFiltParaIndex - 1, scanForward);
002953          WHILE sPar.Scan(paragraph) DO
002954              BEGIN
002955              paragraph.beingFiltered := FALSE;
002956              IF sPar.position = SELF.lastFiltParaIndex THEN
002957                  sPar.Done;
002958              END;
002959          SELF.textSelection.Free;
002960          Free(SELF.filteredStyles);
002961          SUPERSELF.Free;
002962          {$IFC fTrace}EP;{$ENDC}
002963      END;
002964
002965  {$IFC fTextTrace}
002966      PROCEDURE TStyleCmd.Fields(PROCEDURE Field(nameAndType: S255));
002967      BEGIN
002968          SUPERSELF.Fields(Field);
002969          Field('text: TText');
002970          Field('textSelection: TTextSelection');
002971          Field('firstFiltParaIndex: LONGINT');
```

```
002972          Field('lastFiltParaIndex: LONGINT');
002973          Field('filtFirstLP: INTEGER');
002974          Field('filtLastLP: INTEGER');
002975          Field('currFilteredPara: TEditPara');
002976          Field('filteredStyles: TArray');
002977          Field('');
002978      END;
002979  {$ENDC}
002980
002981
002982      PROCEDURE TStyleCmd.Commit;
002983      BEGIN
002984          {$IFC fTrace}BP(10);{$ENDC}
002985          SELF.textSelection.ChangeStyle(SELF.cmdNumber);
002986          {$IFC fTrace}EP;{$ENDC}
002987      END;
002988
002989
002990      PROCEDURE TStyleCmd.FilterAndDo(actualObject: TObject;
002991                                      PROCEDURE DoToObject(filteredObject: TObject));
002992      VAR savedStyles:    TArray;
002993          paragraph:      TEditPara;
002994          typeStyle:      TTypeStyle;
002995          firstLP:        INTEGER;
002996          lastLP:         INTEGER;
002997      BEGIN
002998          {$IFC fTrace}BP(10);{$ENDC}
002999          paragraph := TParaImage(actualObject).paragraph;
003000          IF paragraph.beingFiltered THEN
003001              BEGIN
003002              IF paragraph = SELF.currFilteredPara THEN
003003                  BEGIN
003004                  savedStyles := paragraph.typeStyles;
003005                  paragraph.typeStyles := SELF.filteredStyles;
003006                  END
003007              ELSE
003008                  BEGIN
003009                  IF paragraph = TEditPara(SELF.text.paragraphs.At(SELF.firstFiltParaIndex)) THEN
003010                      firstLP := SELF.filtFirstLP
003011                  ELSE
003012                      firstLP := 0;
003013                  IF paragraph = TEditPara(SELF.text.paragraphs.At(SELF.lastFiltParaIndex)) THEN
003014                      lastLP := SELF.filtLastLP
003015                  ELSE
003016                      lastLP := paragraph.size;
003017                  Free(SELF.filteredStyles);
003018                  savedStyles := TArray(paragraph.typeStyles.Clone(SELF.heap));
003019                  SELF.textSelection.DoChangeStyle(SELF.cmdNumber, paragraph, firstLP, lastLP, typeStyle);
```

```
003020                SELF.currFilteredPara := paragraph;
003021                SELF.filteredStyles := paragraph.typeStyles;
003022                END;
003023
003024            DoToObject(TParaImage(actualObject));
003025
003026            paragraph.typeStyles := savedStyles;
003027            END
003028        ELSE
003029            DoToObject(TParaImage(actualObject));
003030        {$IFC fTrace}EP;{$ENDC}
003031    END;
003032
003033
003034    PROCEDURE TStyleCmd.Perform(cmdPhase: TCmdPhase);
003035    VAR textSelection:  TTextSelection;
003036        selection:      TSelection;
003037        sPar:           TListScanner;
003038        paragraph:      TEditPara;
003039
003040        {Need to filter paragraph before asking about its type styles}
003041        PROCEDURE FindFilteredStyle(obj: TObject);
003042            BEGIN
003043            textSelection.StyleFromContext;
003044            END;
003045
003046    BEGIN
003047        {$IFC fTrace}BP(10);{$ENDC}
003048        selection := SELF.image.view.panel.selection;
003049        WHILE selection.coSelection <> NIL DO
003050            selection := selection.coSelection;
003051
003052        textSelection := TTextSelection(selection);
003053        IF cmdPhase = doPhase THEN
003054            BEGIN
003055            sPar := SELF.text.paragraphs.ScannerFrom(SELF.firstFiltParaIndex - 1, scanForward);
003056            WHILE sPar.Scan(paragraph) DO
003057                BEGIN
003058                paragraph.beingFiltered := TRUE;
003059                IF sPar.position = SELF.lastFiltParaIndex THEN
003060                    sPar.Done;
003061                END;
003062            END;
003063        textSelection.MarkChanged;
003064        textSelection.textImage.text.RecomputeImages;
003065        SELF.FilterAndDo(TParaImage(TEditPara(SELF.text.paragraphs.At(
003066                            SELF.firstFiltParaIndex)).images.First), FindFilteredStyle);
003067        (*textSelection.Invalidate;*)
```

```
003068          {$IFC fTrace}EP;{$ENDC}
003069      END;
003070
003071  {$S SgTxtIni}
003072  END; {METHODS OF TStyleCmd}
003073  {$S SgTxtCld}
003074
003075
003076  METHODS OF TTextCutCopy;
003077
003078      FUNCTION TTextCutCopy.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
003079                                          itsImage: TImage; isCutCmd: BOOLEAN; itsText: TText): TTextCutCopy;
003080      BEGIN
003081          {$IFC fTrace}BP(11);{$ENDC}
003082          IF object = NIL THEN
003083              object := NewObject(heap, THISCLASS);
003084          SELF := TTextCutCopy(TCutCopyCommand.CREATE(object, heap, itsCmdNumber, itsImage, isCutCmd));
003085          WITH SELF DO
003086              BEGIN
003087              text := itsText;
003088              IF itsCmdNumber = uCopy THEN
003089                  BEGIN
003090                  unHiliteBefore[doPhase] := FALSE;
003091                  hiliteAfter[doPhase] := FALSE;
003092                  END;
003093              END;
003094          {$IFC fTrace}EP;{$ENDC}
003095      END;
003096
003097
003098  {$IFC fTextTrace}
003099      PROCEDURE TTextCutCopy.Fields(PROCEDURE Field(nameAndType: S255));
003100      BEGIN
003101          TCutCopyCommand.Fields(Field);
003102          Field('text: TText');
003103          Field('');
003104      END;
003105  {$ENDC}
003106
003107
003108      PROCEDURE TTextCutCopy.DoCutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN;
003109                                           cmdPhase: TCmdPhase);
003110      VAR textSel:        TTextSelection;
003111          insertionPt:    TInsertionPoint;
003112          multiParaSel:   TMultiParaSelection;
003113          heap:           THeap;
003114          firstPara:      TEditPara;
003115          firstLP:        INTEGER;
```

```
003116          panel:          TPanel;
003117          selection:      TSelection;
003118          saveTextSel:    TTextSelection;
003119          firstIndex:     LONGINT;
003120
003121      BEGIN
003122          {$IFC fTrace}BP(10);{$ENDC}
003123          heap := SELF.Heap;
003124          panel := SELF.image.view.panel;
003125          CASE cmdPhase OF
003126              doPhase, redoPhase:
003127                  BEGIN
003128                  selection := panel.selection;
003129
003130                  {we know that the last coSelection must be the textSelection since textSelections
003131                   do not have coSelections}
003132                  WHILE selection.coSelection <> NIL DO
003133                      selection := selection.coSelection;
003134
003135                  textSel := TTextSelection(selection);
003136                  IF (cmdPhase = redoPhase) AND deleteOriginal THEN
003137                      SELF.text.HiliteRange(hOffToOn, textSel.textRange, textSel.isParaSelection);
003138
003139                  textSel.CutCopy(clipSelection, deleteOriginal);
003140                  END;
003141              undoPhase:
003142                  BEGIN
003143                  IF deleteOriginal THEN
003144                      BEGIN
003145                      selection := panel.selection;
003146                      WHILE selection.coSelection <> NIL DO
003147                          selection := selection.coSelection;
003148                      insertionPt := TInsertionPoint(selection);
003149                      firstPara := insertionPt.textRange.firstPara;
003150                      firstLP := insertionPt.textRange.firstLP;
003151                      firstIndex := insertionPt.textRange.firstIndex;
003152
003153                      {get the cut text from the clipboard and insert it back into the text}
003154                      multiParaSel := TMultiParaSelection(clipSelection);
003155                      insertionPt.InsertText(multiParaSel.textImage.text, multiParaSel.isParaSelection,
003156                                                                  multiParaSel.isWordSelection, FALSE);
003157
003158                      IF multiParaSel.isParaSelection THEN
003159                          BEGIN
003160                          WITH insertionPt DO
003161                              IF textRange.firstIndex > 1 THEN
003162                                BEGIN
003163                                textRange.firstIndex := textRange.firstIndex - 1;
```

```
003164                              {$H-}
003165                              textRange.firstPara := TEditPara(SELF.text.paragraphs.At(textRange.firstIndex));
003166                              textRange.firstLP := textRange.firstPara.size;
003167                              {$H+}
003168                              END;
003169                          END
003170                      ELSE IF multiParaSel.isWordSelection THEN
003171                          {don't want to select extra blank generated by insert}
003172                          WITH insertionPt DO
003173                              {$H-}
003174                              IF firstPara.At(firstLP + 1) = ' ' THEN
003175                                  firstLP := firstLP + 1
003176                              ELSE IF textRange.firstPara.At(textRange.firstLP) = ' ' THEN
003177                                  textRange.firstLP := textRange.firstLP - 1;
003178                              {$H+}
003179
003180                      {build the original selection}
003181                      textSel := TTextSelection(selection.FreedAndReplacedBy(
003182                                      insertionPt.textImage.NewTextSelection(
003183                                          firstPara, firstIndex, firstLP, insertionPt.textRange.firstPara,
003184                                          insertionPt.textRange.firstIndex, insertionPt.textRange.firstLP)));
003185                      textSel.isParaSelection := multiParaSel.isParaSelection;
003186
003187                      SELF.text.ChangeSelInOtherPanels(textSel);
003188                      END;
003189                  END;
003190              END;
003191          {$IFC fTrace}EP;{$ENDC}
003192      END;
003193
003194  {$S SgTxtIni}
003195  END; {METHODS OF TTextCutCopy}
003196  {$S SgTxtCld}
003197
003198
003199  METHODS OF TTextPaste;
003200
003201      FUNCTION TTextPaste.CREATE(object: TObject; heap: THeap; itsImage: TImage;
003202                                          itsText: TText): TTextPaste;
003203      VAR range:  TTextRange;
003204      BEGIN
003205          {$IFC fTrace}BP(11);{$ENDC}
003206          IF object = NIL THEN
003207              object := NewObject(heap, THISCLASS);
003208          SELF := TTextPaste(TPasteCommand.CREATE(object, heap, uPaste, itsImage));
003209          {need noSelection since it gets FreedAndReplacedBy'ed}
003210          range := TTextRange.CREATE(NIL, heap, NIL, 0, 0, NIL, 0, 0); {Perform initializes}
003211          WITH SELF DO
```

```
003212              BEGIN
003213              pasteRange := range;
003214              text := itsText;
003215              savedText := NIL;
003216              origIsPara := FALSE;
003217              origIsWord := FALSE;
003218              clipIsPara := FALSE;
003219              END;
003220          {$IFC fTrace}EP;{$ENDC}
003221      END;
003222
003223      PROCEDURE TTextPaste.Free;
003224      BEGIN
003225          {$IFC fTrace}BP(10);{$ENDC}
003226          IF SELF.savedText <> NIL THEN
003227              SELF.savedText.FreeSelf(FALSE);
003228          Free(SELF.pasteRange);
003229          SUPERSELF.Free;
003230          {$IFC fTrace}EP;{$ENDC}
003231      END;
003232
003233  {$IFC fTextTrace}
003234      PROCEDURE TTextPaste.Fields(PROCEDURE Field(nameAndType: S255));
003235      BEGIN
003236          SUPERSELF.Fields(Field);
003237          Field('savedText: TText');
003238          Field('pasteRange: TTextRange');
003239          Field('text: TText');
003240          Field('origIsPara: BOOLEAN');
003241          Field('origIsWord: BOOLEAN');
003242          Field('clipIsPara: BOOLEAN');
003243          Field('');
003244      END;
003245  {$ENDC}
003246
003247      PROCEDURE TTextPaste.Commit;
003248      BEGIN
003249          {$IFC fTrace}BP(10);{$ENDC}
003250          Free(SELF.savedText);
003251          SELF.savedText := NIL;
003252          {$IFC fTrace}EP;{$ENDC}
003253      END;
003254
003255      PROCEDURE TTextPaste.DoPaste(clipSelection: TSelection; pic: PicHandle; cmdPhase: TCmdPhase);
003256      VAR heap:           THeap;
003257          textSel:        TTextSelection;
003258          saveTextSel:    TTextSelection;
003259          insertionPt:    TInsertionPoint;
```

```
003260          insPtBefore:    TInsertionPoint;
003261          insPtAfter:     TInsertionPoint;
003262          text:           TText;
003263          firstPara:      TEditPara;  {bad choice of var names; change later (screws up WITH's)}
003264          firstLP:        INTEGER;
003265          firstIndex:     LONGINT;
003266          panel:          TPanel;
003267          selection:      TSelection;
003268      BEGIN
003269          {$IFC fTrace}BP(10);{$ENDC}
003270          heap := SELF.Heap;
003271          panel := SELF.image.view.panel;
003272          CASE cmdPhase OF
003273              doPhase, redoPhase:
003274                  BEGIN
003275                  IF InClass(clipSelection, TTextSelection) OR (clipBoard.hasUniversalText) THEN
003276                      BEGIN
003277                      selection := panel.selection;
003278                      {we know that the last coSelection must be the textSelection since textSelections
003279                       do not have coSelections}
003280                      WHILE selection.coSelection <> NIL DO
003281                          selection := selection.coSelection;
003282                      textSel := TTextSelection(selection);
003283                      SELF.origIsPara := textSel.isParaSelection;
003284                      SELF.origIsWord := textSel.isWordSelection;
003285                      IF InClass(clipSelection, TTextSelection) THEN
003286                          SELF.clipIsPara := TTextSelection(clipSelection).isParaSelection;
003287
003288                      {delete the selected text, leaving an insertion point}
003289                      text := textSel.DeleteButSave;
003290                      SELF.savedText := text;
003291                      insertionPt := textSel.BecomeInsertionPoint;
003292                      WITH SELF, insertionPt DO
003293                          BEGIN
003294                          pasteRange.firstPara := textRange.firstPara;
003295                          pasteRange.firstIndex := textRange.firstIndex;
003296                          pasteRange.firstLP := textRange.firstLP;
003297                          END;
003298
003299                      insertionPt.FinishPaste(clipSelection, pic);
003300
003301                      WITH SELF, insertionPt DO
003302                          IF clipIsPara THEN
003303                              BEGIN
003304                              {$H-}
003305                              pasteRange.lastIndex := Max(1, textRange.firstIndex - 1);
003306                              pasteRange.lastPara := TEditPara(SELF.text.paragraphs.At(pasteRange.lastIndex));
003307                              pasteRange.lastLP := pasteRange.lastPara.size;
```

```
003308                          {$H+}
003309                          END
003310                      ELSE
003311                          BEGIN
003312                          pasteRange.lastPara := textRange.firstPara;
003313                          pasteRange.lastIndex := textRange.firstIndex;
003314                          pasteRange.lastLP := textRange.firstLP;
003315                          END;
003316
003317                  SELF.text.ChangeSelInOtherPanels(insertionPt);
003318                  END
003319              ELSE
003320                  BEGIN
003321                  panel.selection.CantDoIt;
003322                  SELF.undoable := FALSE;
003323                  END;
003324              END;
003325          undoPhase:
003326              BEGIN
003327              WITH SELF.pasteRange DO
003328                  {$H-}
003329                  textSel := TTextImage(SELF.image).NewTextSelection(firstPara, firstIndex, firstLP,
003330                                                                     lastPara, lastIndex, lastLP);
003331                  {$H+}
003332              textSel.isParaSelection := SELF.clipIsPara;
003333
003334              {user feedback: highlight pasted text}
003335              SELF.text.HiliteRange(hOffToOn, SELF.pasteRange, textSel.isParaSelection);
003336
003337              {get rid of pasted text; can get it from clipboard for redo}
003338              textSel.DeleteAndFree;
003339
003340              insertionPt := textSel.BecomeInsertionPoint;
003341
003342              firstPara := insertionPt.textRange.firstPara;
003343              firstLP := insertionPt.textRange.firstLP;
003344              firstIndex := insertionPt.textRange.firstIndex;
003345
003346              {put back any text that was pasted over}
003347              insertionPt.InsertText(SELF.savedText, SELF.origIsPara, SELF.origIsWord, FALSE);
003348
003349              WITH insertionPt DO
003350                  IF SELF.origIsPara THEN
003351                      BEGIN
003352                      {$H-}
003353                      textRange.firstIndex := Max(1, textRange.firstIndex - 1);
003354                      textRange.firstPara := TEditPara(SELF.text.paragraphs.At(textRange.firstIndex));
003355                      textRange.firstLP := textRange.firstPara.size;
```

```
003356                          {$H+}
003357                          END;
003358
003359                  Free(SELF.savedText);
003360                  SELF.savedText := NIL;
003361
003362                  selection := panel.selection;
003363                  WHILE selection.coSelection <> NIL DO
003364                      selection := selection.coSelection;
003365
003366                  {build original selection (ie before the paste)}
003367                  textSel := TTextSelection(selection.FreedAndReplacedBy(
003368                                  insertionPt.textImage.NewTextSelection(
003369                                      firstPara, firstIndex, firstLP, insertionPt.textRange.firstPara,
003370                                      insertionPt.textRange.firstIndex, insertionPt.textRange.firstLP)));
003371
003372                  textSel.isParaSelection := SELF.origIsPara;
003373                  textSel.isWordSelection := SELF.origIsWord;
003374                  SELF.text.ChangeSelInOtherPanels(textSel);
003375                  insertionPt.Free;
003376                  END;
003377              END;
003378          {$IFC fTrace}EP;{$ENDC}
003379      END;
003380
003381  {$S SgTxtIni}
003382  END; {METHODS OF TTextPaste}
003383  {$S SgTxtHot}
003384
003385  METHODS OF TTypingCmd;
003386
003387      FUNCTION TTypingCmd.CREATE(object: TObject; heap: THeap; itsImage: TImage;
003388                                  itsText: TText): TTypingCmd;
003389      VAR range:  TTextRange;
003390      BEGIN
003391          {$IFC fTrace}BP(11);{$ENDC}
003392          IF object = NIL THEN
003393              object := NewObject(heap, THISCLASS);
003394          SELF := TTypingCmd(TCommand.CREATE(object, heap, uTyping, itsImage, TRUE, revealAll));
003395          range := TTextRange.CREATE(NIL, heap, NIL, 0, 0, NIL, 0, 0); {Perform initializes}
003396          WITH SELF DO
003397              BEGIN
003398              newCharCount := 0;
003399              newParaCount := 0;
003400              text := itsText;
003401              savedText := NIL;
003402              typingRange := range;
003403              otherInsPts := NIL;
```

```
003404              hiliteAfter[doPhase] := FALSE;
003405              END;
003406          {$IFC fTrace}EP;{$ENDC}
003407      END;
003408
003409
003410      PROCEDURE TTypingCmd.Free;
003411      VAR selection: TSelection;
003412      BEGIN
003413          {$IFC fTrace}BP(10);{$ENDC}
003414          Free(SELF.savedText);
003415          selection := SELF.image.view.panel.selection;
003416          WHILE selection.coSelection <> NIL DO
003417              selection := selection.coSelection;
003418
003419          IF InClass(selection, TInsertionPoint) THEN
003420              BEGIN
003421              TInsertionPoint(selection).typingCmd := NIL;
003422              TInsertionPoint(selection).amTyping := FALSE;
003423              END;
003424
003425          SELF.typingRange.Free;
003426          IF SELF.otherInsPts <> NIL THEN
003427              SELF.otherInsPts.FreeObject;
003428          SUPERSELF.Free;
003429          {$IFC fTrace}EP;{$ENDC}
003430      END;
003431
003432  {$S SgTxtCld}
003433  {$IFC fTextTrace}
003434      PROCEDURE TTypingCmd.Fields(PROCEDURE Field(nameAndType: S255));
003435      BEGIN
003436          SUPERSELF.Fields(Field);
003437          Field('');
003438          Field('savedText: TText');
003439          Field('text: TText');
003440          Field('newCharCount: INTEGER');
003441          Field('newParaCount: INTEGER');
003442          Field('typingRange: TTextRange');
003443          Field('otherInsPts: TList');
003444          Field('');
003445      END;
003446  {$ENDC}
003447
003448  {$S SgTxtHot}
003449      PROCEDURE TTypingCmd.Perform(cmdPhase: TCmdPhase);
003450      VAR text:           TText;
003451          insertionPt:    TInsertionPoint;
```

```
003452          selection:      TSelection;
003453          heap:           THeap;
003454          firstPara:      TEditPara;
003455          firstLP:        INTEGER;
003456          textSel:        TTextSelection;
003457          panel:          TPanel;
003458          firstIndex:     LONGINT;
003459          aList:          TList;
003460          typeStyle:      TTypeStyle;
003461
003462          PROCEDURE InstallInsPts(obj: TObject);
003463          VAR selection: TSelection;
003464          BEGIN
003465              selection := TTextImage(obj).view.panel.selection;
003466              WHILE selection.coSelection <> NIL DO
003467                  selection := selection.coSelection;
003468              IF selection <> insertionPt THEN
003469                  aList.InsLast(selection);
003470          END;
003471
003472      BEGIN
003473          {$IFC fTrace}BP(10);{$ENDC}
003474          heap := SELF.Heap;
003475          panel := SELF.image.view.panel;
003476          CASE cmdPhase OF
003477              doPhase, redoPhase:
003478                  BEGIN
003479                  selection := panel.selection;
003480                  WHILE selection.coSelection <> NIL DO
003481                      selection := selection.coSelection;
003482
003483                  textSel := TTextSelection(selection);
003484                  typeStyle := textSel.currTypeStyle;
003485
003486                  {We don't want to delete the entire paragraph if we're typing over it,
003487                   so set isParaSelection to FALSE}
003488                  textSel.isParaSelection := FALSE;
003489
003490  (***** Changed following line 4/27/84 13:07 LSR
003491              BUG: redo of backspace (?) leaves garbage
003492                  deferUpdate := TRUE;
003493  *****)
003494                  deferUpdate := (cmdPhase = doPhase) OR (SELF.savedText <> NIL);
003495
003496                  text := textSel.DeleteButSave;
003497                  insertionPt := textSel.BecomeInsertionPoint;
003498                  WITH SELF.typingRange DO
003499                      BEGIN
```

```
003500                  firstPara := insertionPt.textRange.firstPara;
003501                  firstIndex := insertionPt.textRange.firstIndex;
003502                  firstLP := insertionPt.textRange.firstLP;
003503                  lastPara := firstPara;
003504                  lastIndex := firstIndex;
003505                  lastLP := firstLP;
003506                  END;
003507              IF cmdPhase = doPhase THEN
003508                  BEGIN
003509                  WITH insertionPoint DO
003510                      BEGIN
003511                      amTyping := TRUE;
003512                      typingCmd := SELF;
003513                      IF TFakeTStyle(currTypeStyle) <> TFakeTStyle(typeStyle) THEN
003514                          BEGIN
003515                          styleCmdNumber := -1; {so correct typeStyle will be used}
003516                          currTypeStyle := typeStyle;
003517                          END;
003518                      END;
003519                  {If there is more than one panel displaying this text, then store the insertion points
003520                   of the other panels in a list for quick access while typing}
003521                  IF SELF.text.txtImgList.size > 1 THEN
003522                      BEGIN
003523                      aList := TList.CREATE(NIL, heap, 0);
003524                      SELF.text.txtImgList.Each(InstallInsPt);
003525                      SELF.otherInsPts := aList;
003526                      END;
003527                  insertionPt.textRange.firstPara.StartEdit(insertionPt.textRange.firstPara.GrowSize);
003528                  END
003529              ELSE
003530                  BEGIN
003531                  firstPara := insertionPt.textRange.firstPara;
003532                  firstLP := insertionPt.textRange.firstLP;
003533                  firstIndex := insertionPt.textRange.firstIndex;
003534
003535                  {put back the typed text}
003536                  deferUpdate := FALSE;
003537                  insertionPt.InsertText(SELF.savedText, FALSE, FALSE, FALSE);
003538
003539                  Free(SELF.savedText);
003540
003541                  WITH SELF.typingRange DO
003542                      BEGIN
003543                      lastPara := insertionPt.textRange.firstPara;
003544                      lastIndex := insertionPt.textRange.firstIndex;
003545                      lastLP := insertionPt.textRange.firstLP;
003546                      END;
003547                  {build typed selection}
```

```
003548                  textSel := TTextSelection(insertionPt.FreedAndReplacedBy(
003549                                  insertionPt.textImage.NewTextSelection(
003550                                     firstPara, firstIndex, firstLP, insertionPt.textRange.firstPara,
003551                                     insertionPt.textRange.firstIndex, insertionPt.textRange.firstLP)));
003552
003553                  SELF.text.ChangeSelInOtherPanels(textSel);
003554                  END;
003555
003556              {We always need a valid savedText object, even if no characters were initially typed
003557               over, in case previous characters get backspaced over.  See code in KeyBack }
003558              IF text = NIL THEN
003559                  BEGIN
003560                  text := TText.CREATE(NIL, heap, NIL);
003561                  text.paragraphs.InsLast(
003562                              TTextImage(SELF.image).NewEditPara(0, SELF.typingRange.firstPara.format));
003563                  END;
003564              SELF.savedText := text;
003565              END;
003566          undoPhase:
003567              BEGIN
003568              WITH SELF.typingRange DO
003569                  {$H-}
003570                  textSel := TTextImage(SELF.image).NewTextSelection(firstPara, firstIndex, firstLP,
003571                                                                     lastPara, lastIndex, lastLP);
003572                  {$H+}
003573
003574              {user feedback: highlight typed text}
003575              SELF.text.HiliteRange(hOffToOn, SELF.typingRange, textSel.isParaSelection);
003576
003577              {delete but save typed text}
003578              text := textSel.DeleteButSave;
003579
003580              insertionPt := textSel.BecomeInsertionPoint;
003581
003582              firstPara := insertionPt.textRange.firstPara;
003583              firstLP := insertionPt.textRange.firstLP;
003584              firstIndex := insertionPt.textRange.firstIndex;
003585
003586              {put back any text that was typed over}
003587              insertionPt.InsertText(SELF.savedText, FALSE, FALSE, FALSE);
003588
003589              Free(SELF.savedText);
003590              SELF.savedText := text;
003591
003592              selection := panel.selection;
003593              WHILE selection.coSelection <> NIL DO
003594                  selection := selection.coSelection;
003595
```

```
003596                  {build original selection (before typing)}
003597                  textSel := TTextSelection(selection.FreedAndReplacedBy(
003598                                  insertionPt.textImage.NewTextSelection(
003599                                      firstPara, firstIndex, firstLP, insertionPt.textRange.firstPara,
003600                                      insertionPt.textRange.firstIndex, insertionPt.textRange.firstLP)));
003601
003602                  SELF.text.ChangeSelInOtherPanels(textSel);
003603                  insertionPt.Free;
003604                  END;
003605              END;
003606          {$IFC fTrace}EP;{$ENDC}
003607      END;
003608
003609  {$S SgTxtIni}
003610  END; {METHODS OF TTypingCmd}
003611  {$S SgTxtCld}
003612
003613

End of File -- Lines: 3613 Characters: 131485
```

```
================================================================================================
FILE: "LIBTK/UUNIVTEXT.TEXT"
================================================================================================

000001  {$E+}
000002  {$E ERR1.TEXT}
000003  {-----------------------------------------------------------------------------------------------------------}
000004  {                                          UUniversalText V0.3                                              }
000005  {-----------------------------------------------------------------------------------------------------------}
000006  {$SETC ForOS := TRUE }
000007  {$DECL WithUObject}
000008  {$SETC WithUObject := TRUE}               {Note: TRUE/FALSE status MUST agree with below}
000009
000010
000011
000012  UNIT {$IFC WithUObject}
000013       UTKUniversalText
000014       {$ELSEC}
000015       UUniversalText
000016       {$ENDC};
000017
000018  {$DECL IsIntrinsic}
000019  {$SETC IsIntrinsic := TRUE}
000020
000021  {$SETC WithUObject := TRUE}               {Note: TRUE/FALSE status MUST agree with above}
000022
000023  {$IFC IsIntrinsic}
000024  INTRINSIC;
000025  {$ENDC}
000026
000027  {$IFC NOT WithUObject}
000028      {$SETC LibraryVersion := 30 } { 10 = 1.0 libraries; 13 = 1.3 libraries; 20 = Pepsi, 30 = Spring, etc. }
000029  {$ENDC}
000030
000031  INTERFACE
000032
000033  USES
000034  {$IFC WithUObject}
000035      {$U libtk/UObject}          UObject,
000036      {$SETC fTrce := fTrace}
000037  {$ENDC}
000038      {$U libsm/UnitStd.obj }     UnitStd,
000039      {$U libsm/UnitHz.obj }      UnitHz,
000040  {$IFC NOT WithUObject}
000041      {$U libpl/UClascal}              UClascal,              {Will be in PASLIB in Spring}
000042      {$U libqd/Storage.obj }     Storage,
000043  {$ENDC}
```

```
000044  {$IFC LibraryVersion <= 20}
000045      {$U libfm/FontMgr.obj }     FontMgr,
000046      {$U libqd/QuickDraw.obj }   QuickDraw,
000047  {$ELSEC}
000048      {$U libqd/QuickDraw.obj }   QuickDraw,
000049      {$U libfm/FontMgr.obj }     FontMgr,
000050  {$ENDC}
000051      {$U -#BOOT-SysCall }     Syscall,
000052      {$U libpm/PMDecl.obj }      PMDecl,
000053      {$U libpr/PrStdInfo.obj }   PrStdInfo,
000054      {$U libsu/UnitFmt.obj }     UnitFmt,
000055      {$U libsu/UnitCS.obj }      UnitCS,
000056      {$U libwm/Events.obj }      Events,
000057      {$U libsu/Scrap.obj }       Scrap;
000058
000059  {$DECL fUniversalTextTrace}
000060
000061  {$IFC NOT WithUObject}
000062      {$DECL fDebugMethods}
000063      {$SETC fDebugMethods := FALSE}          {Must be FALSE}
000064
000065      {$DECL fDbgObject}
000066
000067      {$DECL fTrce}
000068      {$SETC fTrce := FALSE}                 {Must be FALSE}
000069
000070      {$SETC fDbgObject := FALSE}             {Set to FALSE for final libraries}
000071  {$ENDC}
000072
000073
000074  {$SETC fUniversalTextTrace := fTrce}        {Normal}
000075
000076  {$DECL PasteTrace}                          {Generates READLN asking for tracing during Write UT}
000077  {$SETC PasteTrace := FALSE}
000078
000079
000080  TYPE
000081
000082  {$IFC NOT WithUObject}
000083
000084      S255 =  STRING[255];
000085      THeap = Ptr;    {alias for THz}
000086      TClass = Ptr;   {alias for TPSliceTable in UClascal}
000087
000088      TCollecHeader =  RECORD
000089          classPtr:       TClass;
000090          size:           LONGINT;    {number of real elements, not counting the hole}
000091          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data; MAXINT if none allowed}
```

```
000092          holeStart:      INTEGER;    {0 = at the beginning, size = at the end; MAXINT = none allowed}
000093          holeSize:       INTEGER;    {measured in MemberBytes units}
000094          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grow the collection by}
000095          END;
000096
000097      TFastString = RECORD            {only access ch[i] when hole is at end & TString is not subclassed}
000098          header:         TCollecHeader;
000099          ch:             PACKED ARRAY[1..32740] OF CHAR;
000100          END;
000101      TPFastString = ^TFastString;
000102      THFastString = ^TPFastString;
000103
000104
000105      TUTObject = SUBCLASS OF NIL
000106
000107          FUNCTION  {TUTObject.}CREATE(heap: THeap): TUTObject; ABSTRACT;
000108          FUNCTION  {TUTObject.}Heap: THeap;                              {which heap it is in}
000109          PROCEDURE {TUTObject.}FreeObject; DEFAULT;                      {frees just the object, not its contents}
000110          PROCEDURE {TUTObject.}Free; DEFAULT;                            {frees the object and its contents}
000111          FUNCTION  {TUTObject.}Class: TClass;
000112          END;
000113
000114      TUTCollection = SUBCLASS OF TUTObject
000115
000116        {Variables}
000117          size:           LONGINT;    {number of real elements, not counting the hole}
000118          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data}
000119          holeStart:      INTEGER;    {0 means hole at the beginning, size means hole at the end}
000120          holeSize:       INTEGER;    {measured in MemberBytes units}
000121          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grow the collection by}
000122
000123          FUNCTION  {TCollection.}CREATE(object: TUTObject; heap: THeap; initialSlack: INTEGER): TUTCollection;
000124          FUNCTION  {TCollection.}AddrMember(i: LONGINT): LONGINT;
000125          FUNCTION  {TCollection.}MemberBytes: INTEGER; ABSTRACT;
000126          PROCEDURE {TCollection.}EditAt(atIndex: LONGINT; deltaMembers: INTEGER);
000127          PROCEDURE {TCollection.}InsManyAt(i: LONGINT; otherCollection: TUTCollection; index, howMany: LONGINT);
000128          PROCEDURE {TCollection.}ResizeColl(membersPlusHole: INTEGER);
000129          PROCEDURE {TCollection.}ShiftColl(afterSrcIndex, afterDstIndex, howMany: INTEGER);
000130          PROCEDURE {TCollection.}StartEdit(withSlack: INTEGER);
000131          PROCEDURE {TCollection.}StopEdit;
000132          END;
000133
000134      TUTArray = SUBCLASS OF TUTCollection
000135
000136          recordBytes: INTEGER;
000137
000138          FUNCTION  {TArray.}CREATE(object: TUTObject; heap: THeap; initialSlack, bytesPerRecord: INTEGER)
000139                                    : TUTArray;
```

```
000140          FUNCTION  {TArray.}MemberBytes: INTEGER; OVERRIDE;
000141          FUNCTION  {TArray.}At(i: LONGINT): Ptr; DEFAULT;
000142          PROCEDURE {TArray.}InsAt(i: LONGINT; pRecord: Ptr); DEFAULT;
000143          PROCEDURE {TArray.}InsLast(pRecord: Ptr);
000144          PROCEDURE {TArray.}DelAll;
000145          PROCEDURE {TArray.}DelAt(i: LONGINT); DEFAULT;
000146          PROCEDURE {TArray.}DelManyAt(i, howMany: LONGINT); DEFAULT;
000147          PROCEDURE {TArray.}PutAt(i: LONGINT; pRecord: Ptr);
000148          END;
000149
000150
000151      TUTString = SUBCLASS OF TUTCollection
000152
000153          FUNCTION  {TString.}CREATE(object: TUTObject; heap: THeap; initialSlack: INTEGER): TUTString;
000154          FUNCTION  {TString.}At(i: LONGINT): CHAR;
000155          FUNCTION  {TString.}MemberBytes: INTEGER; OVERRIDE;
000156          PROCEDURE {TString.}ToPAOCAt(i, howMany: LONGINT; pPackedArrayOfCharacter: Ptr);
000157          PROCEDURE {TString.}InsAt(i: LONGINT; character: CHAR);
000158          PROCEDURE {TString.}InsPAOCAt(i: LONGINT; pPackedArrayOfCharacter: Ptr; howMany: LONGINT);
000159          PROCEDURE {TString.}DelAt(i: LONGINT);
000160          PROCEDURE {TString.}DelManyAt(i, howMany: LONGINT);
000161          PROCEDURE {TString.}DelAll;
000162          END;
000163
000164  {$ENDC}
000165
000166      TEnumLevelOfGranularity = (UTCharacters, UTParagraphs);
000167      TLevelOfGranularity = SET OF TEnumLevelOfGranularity;
000168
000169      TCharDescriptor = RECORD      { character descroptor record }
000170              font:           INTEGER;        { font number }
000171              face:           {$IFC LibraryVersion <= 20}TSeteface{$ELSEC}style{$ENDC};          { formating }
000172              superscript:    -128..127;      { number of bits to superscript }
000173              keepOnSamePage: BOOLEAN;
000174              END;
000175
000176      TTabTypes   = (qLeftTab,    qCenterTab,     qRightTab,      qPeriodTab,     qCommaTab);
000177      TTabFill    = (tNoFill,     tDotFill,       tHyphenFill,    tUnderLineFill);
000178      TParaTypes  = (qLeftPara,   qCenterPara,    qRightPara,     qJustPara);
000179
000180      TTabDescriptor = RECORD
000181              position:           INTEGER;            {Location of the tab}
000182              fillBetweenTabs:    TTabFill;           {Fill character for the tab}
000183              tabType:            TTabTypes;          {Type of tab}
000184              END;
000185
000186      TParaDescriptor = RECORD
000187              paragraphStart:     BOOLEAN;  { TRUE if the beginning of the run is also the beginning of a paragraph}
```

```
000188  {$IFC WithUObject}
000189              additionalChrInParagraph:   INTEGER;
000190  {$ENDC}
000191              firstLineMargin:    INTEGER;            {Left margin of first line}
000192              bodyMargin:         INTEGER;            {Left margin of subsequent lines}
000193              rightMargin:        INTEGER;            {Right margin}
000194              paraLeading:        INTEGER;            {Paragraph leading}
000195              lineSpacing:        0..63;              {Inter-line spacing }
000196  {$IFC WithUObject}
000197              tabTable:           TArray {OF TTabDescriptor}; { table of tabs  }
000198  {$ELSEC}
000199              tabTable:           TUTArray {OF TTabDescriptor}; { table of tabs  }
000200  {$ENDC}
000201              paraType:           TParaTypes;         {Paragraph adjustment }
000202              hasPicture:         BOOLEAN;            {Is there a picture avaible for this paragraph?}
000203              END;
000204
000205
000206  {$IFC WithUObject}
000207      TTKUnivText = SUBCLASS OF TOBJECT
000208  {$ELSEC}
000209      TUnivText   = SUBCLASS OF TUTObject
000210  {$ENDC}
000211          paragraphDescriptor:    TParaDescriptor;
000212          characterDescriptor:    TCharDescriptor;
000213          maxDataSize:            INTEGER;
000214  {$IFC WithUObject}
000215          data:                   TString;
000216  {$ELSEC}
000217          data:                   TUTString;
000218  {$ENDC}
000219          itsOurTString:          BOOLEAN;
000220
000221  {$IFC WithUObject}
000222          FUNCTION  {TTKUnivText.}CREATE(object: TObject;
000223                                         itsHeap: THeap;
000224                                         itsTString: TString;
000225                                         itsDataSize: INTEGER) : TTKUnivText;
000226  {$ELSEC}
000227          FUNCTION  {TUnivText.}CREATE(object: TUTObject;
000228                                         itsHeap: THeap;
000229                                         itsTString: TUTString;
000230                                         itsDataSize: INTEGER) : TUnivText;
000231  {$ENDC}
000232          PROCEDURE {TUnivText.}Free; OVERRIDE;
000233          PROCEDURE {TUnivText.}RunToStream;
000234          PROCEDURE {TUnivText.}StreamToTRun;
000235          PROCEDURE {TUnivText.}TabTableToArgTbd;
```

```
000236          PROCEDURE {TUnivText.}ArgTbdToTabTable;
000237          END;
000238
000239
000240  {$IFC WithUObject}
000241      TTKReadUnivText = SUBCLASS OF TTKUnivText
000242  {$ELSEC}
000243      TReadUnivText   = SUBCLASS OF TUnivText
000244  {$ENDC}
000245
000246  {$IFC WithUObject}
000247          buffer:         TString;
000248  {$ELSEC}
000249          buffer:         TUTString;
000250  {$ENDC}
000251          columnCount:    INTEGER;
000252          dataBeforeTab:  BOOLEAN;
000253
000254  {$IFC WithUObject}
000255          FUNCTION  {TReadUnivText.}CREATE(object: TObject;
000256                                           itsHeap: THeap;
000257                                           itsTString: TString;
000258                                           itsDataSize: INTEGER;
000259                                           LevelOfGranularity: TLevelOfGranularity)
000260                                                                   : TTKReadUnivText;
000261  {$ELSEC}
000262          FUNCTION  {TReadUnivText.}CREATE(object: TUTObject;
000263                                           itsHeap: THeap;
000264                                           itsTString: TUTString;
000265                                           itsDataSize: INTEGER;
000266                                           LevelOfGranularity: TLevelOfGranularity)
000267                                                                   : TReadUnivText;
000268  {$ENDC}
000269
000270          PROCEDURE {TReadUnivText.}Free; OVERRIDE;
000271          PROCEDURE {TReadUnivText.}ReadRun;        { Returns one run of text each time called }
000272          PROCEDURE {TReadUnivText.}Restart;        { Resets the object to read from the begining }
000273
000274          PROCEDURE {TReadUnivText.}ScanTable(VAR rows,
000275                                              tabColumns,
000276                                              tabStopColumns: INTEGER);
000277                                    { Returns number of rows and colums of scrap if a valid table }
000278
000279          FUNCTION  {TReadUnivText.}ReadField(    maxFieldSize: INTEGER;
000280                                              VAR fieldOverflow: BOOLEAN;
000281                                              VAR fieldTerminator: CHAR;
000282                                              VAR tabType: TTabTypes)
000283                                                                   : BOOLEAN;
```

```
000284                                          { Returns one field of text each time called }
000285
000286          FUNCTION  {TReadUnivText.}ReadLine(    maxLineSize: INTEGER;
000287                                          VAR lineOverflow: BOOLEAN;
000288                                          VAR lineTerminator: CHAR)
000289                                                                  : BOOLEAN;
000290                                          { Returns one line of text each time called }
000291          FUNCTION  {TReadUnivText.}GetParaPicture(heap: THeap)
000292                                                                  : PicHandle;
000293                                          { Copies the picture for the current paragraph into heap }
000294          END;
000295
000296
000297  {$IFC WithUObject}
000298      TTKWriteUnivText = SUBCLASS OF TTKUnivText
000299  {$ELSEC}
000300      TWriteUnivText   = SUBCLASS OF TUnivText
000301  {$ENDC}
000302
000303  {$IFC WithUObject}
000304          FUNCTION  {TWriteUnivText.}CREATE(object: TObject;
000305                                          itsHeap: THeap;
000306                                          itsTString: TString;
000307                                          itsDataSize: INTEGER)
000308                                                                  : TTKWriteUnivText;
000309  {$ELSEC}
000310          FUNCTION  {TWriteUnivText.}CREATE(object: TUTObject;
000311                                          itsHeap: THeap;
000312                                          itsTString: TUTString;
000313                                          itsDataSize: INTEGER)
000314                                                                  : TWriteUnivText;
000315  {$ENDC}
000316          PROCEDURE {TWriteUnivText.}FillParagraph;       {Writes one run of text each time called}
000317          END;
000318
000319  {$IFC NOT WithUObject}
000320      FUNCTION  NewUTObject(heap: THeap; itsClass: TClass): TUTObject;
000321  {$ENDC}
000322
000323  {$IFC fUniversalTextTrace}
000324  VAR
000325      fPrintSecrets:  BOOLEAN;
000326  {$ENDC}
000327
000328  IMPLEMENTATION
000329
000330  {$IFC fDbgOk}
000331  {$R+}
```

```
000332  {$ELSEC}
000333  {$R-}
000334  {$ENDC}
000335
000336  {$IFC fSymOk}
000337  {$D+}
000338  {$ELSEC}
000339  {$D-}
000340  {$ENDC}
000341
000342  {$SETC doTraceUT := FALSE}
000343  {$SetC fTraceUT := doTraceUT AND fUniversalTextTrace}
000344
000345  {$IFC WithUObject}
000346  {$S TKUTMain}
000347  {$ELSEC}
000348  {$S UTMain}
000349  {$ENDC}
000350
000351  {$I libut/UUnivText2.text}
000352
000353  {$IFC WithUObject}
000354  {$S TKUTInit}
000355  {$ELSEC}
000356  {$S UTInit}
000357  {$ENDC}
000358
000359  END.

End of File -- Lines: 359 Characters: 13021
```

```
================================================================================================
FILE: "LIBTK/XFER.TEXT"
================================================================================================

000001  ; UNIT XFER; {Copyright 1983, 1984 Apple Computer, Inc.}
000002
000003  ; {changed 01/20/83 2026   Added EqualBytes
000004  ; {changed 01/05/83 2149   Moved several routines to CLASLIB so they can go into PASLIB;
000005  ;                           Here we still have (in SgCLAres):
000006  ;                                LIntDivLint, LIntDivInt, LIntMulInt, LIntAndLInt, LIntOrLInt,
000007  ;                                XferLeft, XferRight, RotatePattern, EnterLisabug}
000008  ; {changed 01/01/83 2000   Added %_JmpTo, %_ExitCaller, %_ExitFunny, %_CallMethod, %_Super;
000009  ;                           Replaced IsJsr by %_NextMethod;
000010  ;                           Deleted XPNewMethod;
000011  ;                           Changed Segment from SgABCres to SgCLAres & added some SgCLAini procedures;
000012  ;                           Added $D information conditioned on DEBUGF flag;
000013  ;                           (Note: SP=A7)}
000014  ; {changed 09/13/83 1115   RELEASE TK7D TO TOOLKIT TEAM}
000015  ; {changed 08/30/83 2000   RELEASE TK7C TO TOOLKIT TEAM}
000016
000017
000018  ;================================================================================================
000019  DEBUGF  .EQU  1                 ; 1 to include $D+ info, 0 to exclude it
000020  ;================================================================================================
000021
000022          .MACRO HEAD
000023            .IF DEBUGF
000024              LINK    A6,#0       ; These two instructions form a slow no-op
000025              MOVE.L  (SP)+,A6
000026            .ENDC
000027          .ENDM
000028
000029          .MACRO TAIL
000030            .IF DEBUGF
000031              UNLK    A6
000032              RTS
000033              .ASCII  %1
000034            .ENDC
000035          .ENDM
000036
000037
000038  ;================================================================================================
000039          .SEG    'SgXFER'
000040  ;================================================================================================
000041
000042
000043          .PROC XFERLEFT
```

```
000044          HEAD
000045
000046  ; PROCEDURE XferLeft(source, dest: TP; nBytes: INTEGER);
000047  ;
000048  ; uses A0,A1,D0,D1
000049  ;
000050
000051          MOVE.L  (SP)+,D1        ; POP RETURN ADDRESS
000052          MOVE.W  (SP)+,D0        ; D0 := NBYTES
000053          MOVE.L  (SP)+,A1        ; A1 := DEST
000054          MOVE.L  (SP)+,A0        ; A0 := SOURCE
000055          MOVE.L  D1,-(SP)        ; PUSH RETURN ADDRESS FOR RTS
000056
000057          SUB.W   #1,D0           ; DECREMENT NBYTES
000058
000059          BLT.S   RTSLEFT         ; NBYTES <= 0, SO EXIT
000060
000061  XFER    MOVE.B  (A0)+,(A1)+
000062          DBF     D0,XFER
000063
000064  RTSLEFT RTS
000065
000066          TAIL    'XFERLEFT'
000067
000068  ;===============================================================================================
000069
000070          .PROC XFERRIGH
000071          HEAD
000072
000073  ; PROCEDURE XferRight(source, dest: TP; nBytes: INTEGER);
000074  ;
000075  ; uses A0,A1,D0,D1
000076  ;
000077
000078          MOVE.L  (SP)+,D1        ; POP RETURN ADDRESS
000079          MOVE.W  (SP)+,D0        ; D0 := NBYTES
000080          MOVE.L  (SP)+,A1        ; A1 := DEST
000081          MOVE.L  (SP)+,A0        ; A0 := SOURCE
000082          MOVE.L  D1,-(SP)        ; PUSH RETURN ADDRESS FOR RTS
000083
000084          TST.W   D0              ; TEST NBYTES
000085          BLE.S   RTSRIGH         ; NBYTES <= 0, SO EXIT
000086
000087          ADDA.W  D0,A0           ; START AT RIGHT END
000088          ADDA.W  D0,A1
000089
000090          SUB.W   #1,D0           ; DECREMENT NBYTES
000091
```

```
000092  XFER    MOVE.B  -(A0),-(A1)
000093          DBF     D0,XFER
000094
000095  RTSRIGH RTS
000096
000097          TAIL    'XFERRIGH'
000098
000099  ;===============================================================================================
000100
000101          .PROC EqualBytes
000102          HEAD
000103
000104  ; PROCEDURE EqualBytes(source, dest: TP; nBytes: INTEGER);
000105  ;
000106  ; uses A0,A1,D0,D1
000107  ;
000108
000109          MOVE.L  (SP)+,D1        ; POP RETURN ADDRESS
000110          MOVE.W  (SP)+,D0        ; D0 := NBYTES
000111          MOVE.L  (SP)+,A1        ; A1 := DEST
000112          MOVE.L  (SP)+,A0        ; A0 := SOURCE
000113          MOVE.L  D1,-(SP)        ; PUSH RETURN ADDRESS FOR RTS
000114
000115          MOVE.B  #1,4(SP)        ; RETURN TRUE UNLESS PROVEN UNEQUAL
000116
000117          SUB.W   #1,D0           ; DECREMENT NBYTES
000118
000119          BLT.S   RTSEQUL         ; NBYTES <= 0, SO EXIT
000120
000121  XFER    MOVE.B  (A0)+,D1
000122          CMP.B   (A1)+,D1
000123          BNE     UNEQUL
000124          DBF     D0,XFER
000125
000126  RTSEQUL RTS
000127
000128  UNEQUL  CLR.B   4(SP)           ; RETURN FALSE
000129          RTS
000130
000131          TAIL    'EQUALBYT'
000132
000133  ;===============================================================================================
000134
000135
000136          .PROC ROTATEPA
000137          HEAD
000138
000139  ; PROCEDURE RotatePattern(pInPat, pOutPat: ^Pattern; dh, dv: LONGINT);
```

```
000140  ;
000141  ; uses A0-A2,D0-D4
000142  ;
000143
000144          MOVEM.L (SP)+,D0-D2/A0-A1  ; D0 := RETURN ADDRESS, D1 := dv; D2 := dh, A0 := pOutPat, A1 := pInPat
000145          MOVE.L  D0,-(SP)           ; PUSH RETURN ADDRESS FOR RTS
000146          MOVEM.L A2/D3-D4,-(SP)     ; Save A2,D3,D4
000147
000148          AND.L   #7,D2              ; dh := dh MOD 8
000149
000150  ; ***** FOR D3 := 7 DOWNTO 0 DO *****
000151
000152          MOVE.W  #7,D3           ; Loop count
000153
000154  ; *****    BEGIN *****
000155
000156  RLOOP   MOVE.B  $00(A1,D3.W),D0 ; D0 := next byte in inPat
000157
000158          ROL.B   D2,D0           ; Rotate byte in D0 left by D2 (dh)
000159
000160          MOVE.W  D3,D4
000161          SUB.W   D1,D4
000162          AND.W   #7,D4           ; D4 := (D3 - dv) MOD 8
000163
000164          MOVE.B  D0,$00(A0,D4.W) ; next byte in outPat := D0
000165
000166          DBF     D3,RLOOP
000167
000168  ; *****    END; *****
000169
000170          MOVEM.L (SP)+,A2/D3-D4  ; Restore A2,D3,D4
000171
000172          RTS
000173
000174          TAIL    'ROTATEPA'
000175
000176  ;===============================================================================================
000177
000178          .FUNC LINTDIVL
000179          HEAD
000180
000181  ; FUNCTION  LIntDivLint(i, j: LONGINT): LONGINT;
000182  ;
000183  ; uses A0,D0,D1
000184  ;
000185
000186          .REF    LD
000187
```

```
000188          MOVE.L  (SP)+,A0        ; Return address
000189
000190          MOVE.L  (SP)+,D0        ; D0 := j
000191          MOVE.L  (SP)+,D1        ; D1 := i
000192
000193  CHEK    CMP.L   #32767,D0       ; Is j too long to use LIntDivInt?
000194          BGT     TOOLONG         ; Too big
000195          CMP.L   #-32768,D0
000196          BLT     TOOLONG         ; Too small
000197          JMP     LD              ; Can't BGT LD in this assembler
000198
000199  TOOLONG ASR.L   #1,D0
000200          ASR.L   #1,D1
000201          JMP     CHEK
000202
000203          TAIL    'LINTDIVL'
000204
000205  ;=======================================================================================
000206
000207          .FUNC LINTDIVI
000208          HEAD
000209
000210  ; FUNCTION  LIntDivInt(i: LONGINT; j: INTEGER): LONGINT;
000211  ;
000212  ; uses A0,D0,D1,D2
000213  ;
000214
000215          .DEF    LD
000216
000217          MOVE.L  (SP)+,A0        ; Return address
000218
000219          MOVE.W  (SP)+,D0        ; D0 := j
000220          MOVE.L  (SP)+,D1        ; D1 := i
000221
000222  LD      CMP.W   #1,D0           ; IF j = 1, return i
000223          BEQ     DV1
000224
000225          MOVE.L  D1,-(SP)        ; Push i as LONGINT
000226          MOVE.W  D0,-(SP)        ; Push j as INTEGER
000227
000228          CLR.L   D1
000229
000230          TST.W   (SP)            ; If j is negative, negate both
000231          BGE     JPOS
000232          NEG.W   (SP)            ; negate j
000233          NEG.L   2(SP)           ; negate i
000234
000235  JPOS    TST.L   2(SP)           ; If i is negative, negate it but remember it was
```

```
000236          SMI     D2              ; D2 := (i < 0)
000237          BGE     IPOS
000238          NEG.L   2(SP)           ; negate i
000239
000240  IPOS    MOVE.W  2(SP),D1        ; Divide MSW of i by j
000241          DIVU    (SP),D1
000242
000243          MOVE.L  D1,D0           ; Remainder becomes MSW of next Divide
000244          MOVE.W  4(SP),D0        ; Divide ((preceding remainder) concat (LSW of i)) by j
000245          DIVU    (SP)+,D0        ; Pop j at the same time
000246
000247          SWAP    D1              ; Quotient of first divide is MSW of result
000248          MOVE.W  D0,D1           ; Quotient of second divide is LSW of result
000249
000250          TST.B   D2              ; Was i negative?
000251          BEQ     DUN
000252          NEG.L   D1
000253
000254  DUN     ADD.L   #4,SP           ; Popeye
000255
000256  DV1     MOVE.L  D1,(SP)         ; Store function result
000257
000258          JMP     (A0)            ; Return
000259
000260          TAIL    'LINTDIVI'
000261
000262  ;===============================================================================================
000263
000264          .FUNC LINTMULI
000265          HEAD
000266
000267  ; FUNCTION  LIntMulInt(i: LONGINT; j: INTEGER): LONGINT;
000268  ;
000269  ; uses A0,D0,D1,D2
000270  ;
000271
000272          MOVE.L  (SP)+,A0        ; Return address
000273
000274          MOVE.W  (SP)+,D1        ; D1 := j
000275          MOVE.L  (SP)+,D0        ; D0 := i
000276
000277          CMP.W   #1,D1           ; IF j = 1, return i
000278          BEQ     MU1
000279
000280          MOVE.L  D2,-(SP)        ; Save D2
000281
000282          MOVE.W  D0,D2           ; D2 := LSW of I
000283          SWAP    D0              ; D0 := MSW of I
```

```
000284
000285          MULU    D1,D0           ; D0 := D0 * j
000286          LSL.L   #8,D0
000287          LSL.L   #8,D0
000288          MULU    D1,D2
000289          ADD.L   D2,D0           ; D0 := product
000290
000291          MOVE.L  (SP)+,D2        ; restore D2
000292
000293  MU1     MOVE.L  D0,(SP)         ; Store function result
000294
000295          JMP     (A0)            ; Return
000296
000297          TAIL    'LINTMULI'
000298
000299  ;=============================================================================================
000300
000301          .FUNC LINTANDL
000302          HEAD
000303
000304  ; FUNCTION  LIntAndLInt(i, j: LONGINT): LONGINT;
000305  ;
000306  ; uses A0,D0
000307  ;
000308
000309          MOVE.L  (SP)+,A0        ; Return address
000310
000311          MOVE.L  (SP)+,D0        ; D0 := j
000312          AND.L   (SP)+,D0        ; D0 := i AND j
000313
000314          MOVE.L  D0,(SP)         ; Store function result
000315
000316          JMP     (A0)            ; Return
000317
000318          TAIL    'LINTANDL'
000319
000320  ;=============================================================================================
000321
000322          .FUNC LINTORLI
000323          HEAD
000324
000325  ; FUNCTION  LIntOrLInt(i, j: LONGINT): LONGINT;
000326  ;
000327  ; uses A0,D0
000328  ;
000329
000330          MOVE.L  (SP)+,A0        ; Return address
000331
```

```
000332          MOVE.L  (SP)+,D0        ; D0 := j
000333          OR.L    (SP)+,D0        ; D0 := i OR j
000334
000335          MOVE.L  D0,(SP)         ; Store function result
000336
000337          JMP     (A0)            ; Return
000338
000339          TAIL    'LINTORLI'
000340
000341  ;=======================================================================================
000342
000343          .FUNC LINTXORL
000344          HEAD
000345
000346  ; FUNCTION  LIntXorLInt(i, j: LONGINT): LONGINT;
000347  ;
000348  ; uses A0,D0, D1
000349  ;
000350
000351          MOVE.L  (SP)+,A0        ; Return address
000352
000353          MOVE.L  (SP)+,D0        ; D0 := j
000354          MOVE.L  (SP)+,D1        ; D1 := i
000355          EOR.L   D1,D0           ; D0 := i XOR j
000356
000357          MOVE.L  D0,(SP)         ; Store function result
000358
000359          JMP     (A0)            ; Return
000360
000361          TAIL    'LINTXORL'
000362
000363  ;=======================================================================================
000364
000365          .PROC ENTERLIS
000366          HEAD
000367
000368  ; PROCEDURE EnterLisabug;
000369
000370          TRAP    #0
000371          RTS
000372
000373          TAIL    'ENTERLIS'
000374
000375  ;=======================================================================================
000376
000377
000378          .END
000379
```

```
000380
000381

End of File -- Lines: 381 Characters: 9390
```

```
================================================================================================
FILE: "LIBUT/UUNIVTEXT2.TEXT"
================================================================================================

000001  {$SETC PasteTrace := PasteTrace AND fUniversalTextTrace}
000002
000003  CONST
000004      magicTabMax= itbdLst;   {The maximum number of tabstops on a ruler allowed by LisaWrite and its ilk}
000005      maxBacking = 12;        {Maximum number of chars saved for backing up the lpLim during Write UT}
000006
000007  TYPE
000008      TPtrToolKitUT = ^ToolKitUT;
000009      ToolKitUT =     Tcs;
000010      TSavedPara =    RECORD
000011          firstLp:        TLp;
000012          theArce:        TArce;
000013          theArpe:        TArpe;
000014  {$IFC WithUObject}
000015          theText:        TString;
000016  {$ELSEC}
000017          theText:        TUTString;
000018  {$ENDC}
000019                      END;
000020      PSavedPara =    ^TSavedPara;
000021      HSavedPara =    ^PSavedPara;
000022
000023  {$IFC NOT WithUObject}
000024      Byte =          -128..127;
000025      TpLONGINT =     ^LONGINT;
000026      TPByte =        ^Byte;
000027  {$ENDC}
000028
000029  {private types not used in the Toolkit; used in place of the Toolkit's type coercion to a
000030          Handle, since a Handle outside of the Toolkit is a double-indeirect pointer to a byte}
000031      UTpLongint  = ^LONGINT;
000032      UTppLongint = ^UTpLongint;
000033
000034    { Carefull, carefull, carefull here kids. Since I can't have private fields and/or methods in my classes
000035      inorder to resolve a few types I am forced to do this thing to keep you innocents from having to
000036      include an ugly list of units. Only one instance of these variabes exists ever! Therefore I can only
000037      do things one at a time.}
000038
000039      TSecretThings = RECORD
000040              streamArrayIndex:   Byte;
000041              lpd:                TALpd;
000042              achad:              TAchad;
000043              END;
```

```
000044
000045  VAR
000046  {$IFC WithUObject}
000047      activeStream:   TTKWriteUnivText;
000048  {$ELSEC}
000049      activeStream:   TWriteUnivText;
000050  {$ENDC}
000051      secrets:        TSecretThings;
000052      currentLpd:     TLpd;
000053      dataIndex:      INTEGER;
000054      dataLp:         TLp;
000055      savedPara:      ARRAY [1..maxBacking] OF HSavedPara;
000056      nOfSavedPara:   0..maxBacking;
000057  {$IFC WithUObject}
000058      theData:        TString;
000059  {$ELSEC}
000060      theData:        TUTString;
000061  {$ENDC}
000062
000063  {$IFC NOT WithUObject}
000064
000065  {The following is a µToolKit to avoid including lots of code that is not used by non-ToolKit applications.}
000066
000067  {$IFC WithUObject}
000068  {$S TKUTMain}
000069  {$ELSEC}
000070  {$S UTMain}
000071  {$ENDC}
000072  {-----------------------------------------------------------------------------------------------------------}
000073  PROCEDURE UTXferRight(source, dest: Ptr; nBytes: INTEGER); EXTERNAL;
000074  PROCEDURE UTXferLeft(source, dest: Ptr; nBytes: INTEGER); EXTERNAL;
000075  {-----------------------------------------------------------------------------------------------------------}
000076
000077  {$IFC WithUObject}
000078  {$S TKUTMain}
000079  {$ELSEC}
000080  {$S UTMain}
000081  {$ENDC}
000082  {-----------------------------------------------------------------------------------------------------------}
000083  FUNCTION  Min(i, j: LONGINT): LONGINT;
000084  {-----------------------------------------------------------------------------------------------------------}
000085  BEGIN
000086  {$IFC fTraceUT}    LogCall;     {$ENDC}
000087      IF i < j THEN
000088          Min := i
000089      ELSE
000090          Min := j;
000091  END;
```

```
000092
000093
000094  {$IFC WithUObject}
000095  {$S TKUTMain}
000096  {$ELSEC}
000097  {$S UTMain}
000098  {$ENDC}
000099  {-------------------------------------------------------------------------------------------------------}
000100  FUNCTION  Max(i, j: LONGINT): LONGINT;
000101  {-------------------------------------------------------------------------------------------------------}
000102  BEGIN
000103  {$IFC fTraceUT}    LogCall;     {$ENDC}
000104      IF i > j THEN
000105          Max := i
000106      ELSE
000107          Max := j;
000108  END;
000109
000110
000111  {$IFC WithUObject}
000112  {$S TKUTInit}
000113  {$ELSEC}
000114  {$S UTInit}
000115  {$ENDC}
000116  {-------------------------------------------------------------------------------------------------------}
000117  PROCEDURE ABCBreak(s: S255; errCode: LONGINT);
000118  {-------------------------------------------------------------------------------------------------------}
000119  BEGIN
000120  {$IFC fTraceUT}    LogCall;     {$ENDC}
000121      {$IFC fDbgObject}
000122      WriteLn;
000123      Write(CHR(7), s);       {Beep}
000124      IF errCode <> 0 THEN
000125          Write(': ', errCode:1);
000126      WriteLn;
000127      {$ENDC}
000128      HALT;
000129  END;
000130
000131
000132  {$IFC WithUObject}
000133  {$S TKUTInit}
000134  {$ELSEC}
000135  {$S UTInit}
000136  {$ENDC}
000137  {-------------------------------------------------------------------------------------------------------}
000138  PROCEDURE  SetCp(object: TUTObject; itsClass: TClass);
000139  {-------------------------------------------------------------------------------------------------------}
```

```
000140      VAR index:  INTEGER;
000141  BEGIN
000142  {$IFC fTraceUT}    LogCall;     {$ENDC}
000143      UTppLongint(object)^^ := ORD(itsClass);     {Install slice table pointer}
000144      index := CiOfCp(TPSliceTable(itsClass));    {Determine its class index}
000145      IF index < 256 THEN                         {If it will fit in a byte, store it...}
000146          TPByte(UTppLongint(object)^)^ := index; {...to speed version conversion (cf ConvertHeap: FindClasses)}
000147  END;
000148
000149
000150  {$IFC WithUObject}
000151  {$S TKUTMain}
000152  {$ELSEC}
000153  {$S UTMain}
000154  {$ENDC}
000155  {-------------------------------------------------------------------------------------------------------}
000156  FUNCTION  NewDynObject(heap: THeap; itsClass: TClass; dynBytes: INTEGER): TUTObject;
000157  {-------------------------------------------------------------------------------------------------------}
000158      VAR nBytes: INTEGER;
000159          object: TUTObject;
000160  BEGIN
000161  {$IFC fTraceUT}    LogCall;     {$ENDC}
000162      nBytes := SizeOfCp(TPSliceTable(itsClass)) + dynBytes;
000163      object := POINTER(ORD(HAllocate(THz(heap), nBytes)));    {TUTObject() won't work until after SetCp}
000164      IF ORD(object) = ORD(hNIL) THEN
000165         ABCBreak('NewObject: Heap full, can''t make an object of size', nBytes);
000166      SetCp(object, itsClass);
000167      NewDynObject := object;
000168  END;
000169
000170
000171  {$IFC WithUObject}
000172  {$S TKUTMain}
000173  {$ELSEC}
000174  {$S UTMain}
000175  {$ENDC}
000176  {-------------------------------------------------------------------------------------------------------}
000177  FUNCTION  NewUTObject(heap: THeap; itsClass: TClass): TUTObject;
000178  {-------------------------------------------------------------------------------------------------------}
000179  BEGIN
000180  {$IFC fTraceUT}    LogCall;     {$ENDC}
000181      NewUTObject := NewDynObject(heap, itsClass, 0);
000182  END;
000183
000184
000185  {$IFC WithUObject}
000186  {$S TKUTMain}
000187  {$ELSEC}
```

```
000188  {$S UTMain}
000189  {$ENDC}
000190  {-----------------------------------------------------------------------------------------------------}
000191  PROCEDURE ResizeDynObject(object: TUTObject; newTotalBytes: INTEGER);
000192  {-----------------------------------------------------------------------------------------------------}
000193      VAR i: INTEGER;
000194  BEGIN
000195  {$IFC fTraceUT}    LogCall;     {$ENDC}
000196      IF (newTotalBytes < 0) OR (newTotalBytes > (MAXINT-20)) THEN
000197          ABCBreak('New size must lie between 0 and 32K-20, not', newTotalBytes);
000198      ChangeSizeH(THz(object.Heap), TH(object), newTotalBytes);
000199      IF CbDataOfH(THz(object.Heap), TH(object)) < newTotalBytes THEN
000200         ABCBreak('ResizeDynObject: Heap full, size can''t change to', newTotalBytes);
000201  END;
000202
000203
000204  {$IFC WithUObject}
000205  {$S TKUTMain}
000206  {$ELSEC}
000207  {$S UTMain}
000208  {$ENDC}
000209  {-----------------------------------------------------------------------------------------------------}
000210  FUNCTION ClassPtr(hndl: UTppLongint): TClass;
000211  {-----------------------------------------------------------------------------------------------------}
000212      VAR stp: RECORD
000213                  CASE INTEGER OF
000214                      1: (asLong: LONGINT);
000215                      2: (asBytes: PACKED ARRAY [0..3] OF TByte);
000216                      3: (asClass: TClass);
000217                      END;
000218  BEGIN
000219  {$IFC fTraceUT}    LogCall;     {$ENDC}
000220      stp.asLong := hndl^^;
000221      stp.asBytes[0] := 0;
000222      ClassPtr := stp.asClass;
000223  END;
000224
000225
000226  {$IFC WithUObject}
000227  {$S TKUTMain}
000228  {$ELSEC}
000229  {$S UTMain}
000230  {$ENDC}
000231  {-----------------------------------------------------------------------------------------------------}
000232  FUNCTION  SizeOfClass(class: TClass): INTEGER;
000233  {-----------------------------------------------------------------------------------------------------}
000234  BEGIN
000235  {$IFC fTraceUT}    LogCall;     {$ENDC}
```

```
000236      SizeOfClass := SizeOfCp(TPSliceTable(class));
000237  END;
000238
000239
000240  {$IFC WithUObject}
000241  {$S TKUTInit}
000242  {$ELSEC}
000243  {$S UTInit}
000244  {$ENDC}
000245  {-----------------------------------------------------------------------------------------------------}
000246  PROCEDURE InitObject;
000247  {-----------------------------------------------------------------------------------------------------}
000248  BEGIN
000249  {$IFC fTraceUT}    LogCall;     {$ENDC}
000250      {Do very little for the time beeing}
000251  END;
000252
000253
000254  {$IFC WithUObject}
000255  {$S TKUTInit}
000256  {$ELSEC}
000257  {$S UTInit}
000258  {$ENDC}
000259  {-----------------------------------------------------------------------------------------------------}
000260  PROCEDURE ClascalError(error: INTEGER); {called with error = 0 after successful Clascal initialization}
000261  {-----------------------------------------------------------------------------------------------------}
000262  BEGIN
000263  {$IFC fTraceUT}    LogCall;     {$ENDC}
000264      IF error > 0 THEN
000265          ABCBreak('Some kind of Clascal error', error);
000266  END;
000267
000268
000269  METHODS OF TUTObject;
000270
000271  {$IFC WithUObject}
000272  {$S TKUTMain}
000273  {$ELSEC}
000274  {$S UTMain}
000275  {$ENDC}
000276      {-----------------------------------------------------------------------------------------------------}
000277      PROCEDURE {TObject.}Free;
000278      {-----------------------------------------------------------------------------------------------------}
000279      BEGIN
000280  {$IFC fTraceUT}    LogCall;     {$ENDC}
000281          SELF.FreeObject;
000282      END;
000283
```

```
000284
000285  {$IFC WithUObject}
000286  {$S TKUTMain}
000287  {$ELSEC}
000288  {$S UTMain}
000289  {$ENDC}
000290      {-------------------------------------------------------------------------------------------------}
000291      PROCEDURE {TObject.}FreeObject;
000292      {-------------------------------------------------------------------------------------------------}
000293          VAR heap:       THeap;
000294      BEGIN
000295  {$IFC fTraceUT}    LogCall;     {$ENDC}
000296          heap := SELF.Heap;
000297          FreeH(THz(heap), TH(SELF));
000298      END;
000299
000300
000301  {$IFC WithUObject}
000302  {$S TKUTMain}
000303  {$ELSEC}
000304  {$S UTMain}
000305  {$ENDC}
000306      {-------------------------------------------------------------------------------------------------}
000307      FUNCTION  {TObject.}Heap: THeap;
000308      {-------------------------------------------------------------------------------------------------}
000309      BEGIN
000310  {$IFC fTraceUT}    LogCall;     {$ENDC}
000311          Heap := THeap(HzFromH(TH(SELF)));
000312      END;
000313
000314
000315  {$IFC WithUObject}
000316  {$S TKUTMain}
000317  {$ELSEC}
000318  {$S UTMain}
000319  {$ENDC}
000320      {-------------------------------------------------------------------------------------------------}
000321      FUNCTION  {TObject.}Class: TClass;
000322      {-------------------------------------------------------------------------------------------------}
000323      BEGIN
000324  {$IFC fTraceUT}    LogCall;     {$ENDC}
000325          Class := ClassPtr(UTppLongint(SELF));
000326      END;
000327
000328
000329  {$IFC WithUObject}
000330  {$S TKUTInit}
000331  {$ELSEC}
```

```
000332  {$S UTInit}
000333  {$ENDC}
000334  BEGIN {Class Initialization}
000335
000336  {$IFC fTraceUT}    LogCall;     {$ENDC}
000337      InitClascal(ClascalError);   {Provide an error routine in case of errors in Clascal run-time support}
000338      InitObject;                  {Do remaining initialization}
000339  END;
000340
000341
000342  METHODS OF TUTCollection;
000343
000344  {$IFC WithUObject}
000345  {$S TKUTMain}
000346  {$ELSEC}
000347  {$S UTMain}
000348  {$ENDC}
000349      {-------------------------------------------------------------------------------------------------}
000350      FUNCTION  {TCollection.}CREATE(object: TUTObject; heap: THeap; initialSlack: INTEGER): TUTCollection;
000351      {-------------------------------------------------------------------------------------------------}
000352      BEGIN
000353  {$IFC fTraceUT}    LogCall;     {$ENDC}
000354          IF object = NIL THEN
000355              ABCBreak('TUTCollection.CREATE must be passed an already-allocated object by a subclass CREATE', 0);
000356          SELF := TUTCollection(object);
000357          WITH SELF DO
000358              BEGIN
000359              size := 0;
000360        {$H-} dynStart := SizeOfClass(SELF.Class); {$H+}
000361              holeStart := 0;
000362              holeSize := initialSlack;
000363              holeStd := 0;
000364              END;
000365      END;
000366
000367
000368  {$IFC WithUObject}
000369  {$S TKUTMain}
000370  {$ELSEC}
000371  {$S UTMain}
000372  {$ENDC}
000373      {-------------------------------------------------------------------------------------------------}
000374      FUNCTION  {TCollection.}AddrMember(i: LONGINT): LONGINT;
000375      {-------------------------------------------------------------------------------------------------}
000376      BEGIN
000377  {$IFC fTraceUT}    LogCall;     {$ENDC}
000378          IF i > SELF.holeStart THEN
000379              i := i + SELF.holeSize;
```

```
000380          AddrMember := TpLONGINT(SELF)^ + SELF.dynStart + (SELF.MemberBytes * (i - 1));
000381      END;
000382
000383
000384  {$IFC WithUObject}
000385  {$S TKUTMain}
000386  {$ELSEC}
000387  {$S UTMain}
000388  {$ENDC}
000389      {-------------------------------------------------------------------------------------------------}
000390      PROCEDURE {TCollection.}EditAt(atIndex: LONGINT; deltaMembers: INTEGER);
000391      {-------------------------------------------------------------------------------------------------}
000392          VAR oldHoSize:      INTEGER;
000393              newHoSize:      INTEGER;
000394              oldHoStart:     INTEGER;
000395              newHoStart:     INTEGER;
000396              maxHoStart:     INTEGER;
000397              minHoStart:     INTEGER;
000398              size:           INTEGER;
000399              b:              0..1;
000400      BEGIN                                               {Removes any hole it creates unless holdStd <> 0}
000401  {$IFC fTraceUT}    LogCall;     {$ENDC}
000402          oldHoSize := SELF.holeSize;
000403          oldHoStart := SELF.holeStart;
000404
000405          IF (deltaMembers < 0) AND ((oldHoStart + 1) = atIndex) THEN {the hole is right before the deletion}
000406              SELF.holeStart := oldHoStart - deltaMembers
000407          ELSE
000408              BEGIN
000409              newHoStart := atIndex - 1 - Min(deltaMembers, 0);
000410              IF (deltaMembers > oldHoSize) OR (newHoStart <> oldHoStart) THEN
000411                  BEGIN
000412                  maxHoStart := Max(oldHoStart, newHoStart);
000413                  newHoSize := Max(oldHoSize, deltaMembers);
000414
000415                  IF newHoSize > oldHoSize THEN
000416                      BEGIN
000417                      size := SELF.size;
000418                      newHoSize := Max(newHoSize, SELF.holeStd);
000419                      SELF.ResizeColl(size + newHoSize);
000420                      SELF.ShiftColl(maxHoStart + oldHoSize, maxHoStart + newHoSize, size - maxHoStart);
000421                      END;
000422
000423                  IF newHoStart <> oldHoStart THEN
000424                      BEGIN
000425                      b := ORD(newHoStart > oldHoStart);    {1 if hole is moving right and data is moving left}
000426                      minHoStart := Min(oldHoStart, newHoStart);
000427                      SELF.ShiftColl(minHoStart + oldHoSize*b, minHoStart + newHoSize*(1-b), maxHoStart - minHoStart);
```

```
000428                  END;
000429
000430              SELF.holeStart := newHoStart;
000431              SELF.holeSize := newHoSize;
000432              END;
000433          END;
000434
000435      WITH SELF DO
000436          BEGIN
000437          size := size + deltaMembers;
000438          holeSize := holeSize - deltaMembers;
000439          holeStart := holeStart + deltaMembers;
000440          IF oldHoSize = 0 THEN
000441              IF holeStd = 0 THEN
000442                  IF holeSize > 0 THEN
000443              {$H-}   SELF.StopEdit;  {$H+}
000444          END;
000445  END;
000446
000447
000448 {$IFC WithUObject}
000449 {$S TKUTMain}
000450 {$ELSEC}
000451 {$S UTMain}
000452 {$ENDC}
000453      {-------------------------------------------------------------------------------------------------}
000454      PROCEDURE {TCollection.}InsManyAt(i: LONGINT; otherCollection: TUTCollection; index, howMany: LONGINT);
000455      {-------------------------------------------------------------------------------------------------}
000456          VAR memberBytes:    INTEGER;
000457              beforeHole:     INTEGER;
000458              srcAddr:        LONGINT;
000459              dstAddr:        LONGINT;
000460              j:              INTEGER;
000461              offset:         INTEGER;
000462      BEGIN                                               {Stops edit if it wasn't explicitly started}
000463 {$IFC fTraceUT}    LogCall;     {$ENDC}
000464          memberBytes := SELF.memberBytes;
000465
000466          SELF.EditAt(i, howMany);
000467
000468          IF howMany > 0 THEN
000469              IF otherCollection.Class = SELF.Class THEN          {Can do it with at most two Xfers}
000470                  BEGIN
000471                  beforeHole := Min(howMany, otherCollection.holeStart + 1 - index);
000472
000473                  srcAddr := otherCollection.AddrMember(index);
000474                  dstAddr := SELF.AddrMember(i);
000475                  IF beforeHole > 0 THEN
```

```
000476                  BEGIN
000477                  UTXferLeft(Ptr(srcAddr), Ptr(dstAddr), beforeHole * memberBytes);
000478
000479                  IF beforeHole < howMany THEN
000480                      BEGIN
000481                      srcAddr := srcAddr + (beforeHole + otherCollection.holeSize) * memberBytes;
000482                      dstAddr := dstAddr + beforeHole * memberBytes;
000483                      UTXferLeft(Ptr(srcAddr), Ptr(dstAddr), (howMany - beforeHole) * memberBytes);
000484                      END;
000485                  END
000486              ELSE
000487                  UTXferLeft(Ptr(srcAddr), Ptr(dstAddr), howMany * memberBytes);
000488              END
000489          ELSE                                                {Must Xfer each member separately}
000490              BEGIN
000491              offset := SELF.dynStart + (i - 1) * memberBytes;  {AddrMember may even compact for all we know}
000492              FOR j := 1 TO howMany DO
000493                  BEGIN
000494                  UTXferLeft(Ptr(otherCollection.AddrMember(j)), Ptr(TpLONGINT(SELF)^ + offset), memberBytes);
000495                  offset := offset + memberBytes;
000496                  END;
000497              END;
000498      END;
000499
000500
000501  {$IFC WithUObject}
000502  {$S TKUTMain}
000503  {$ELSEC}
000504  {$S UTMain}
000505  {$ENDC}
000506      {-------------------------------------------------------------------------------------------------}
000507      PROCEDURE {TCollection.}ResizeColl(membersPlusHole: INTEGER);
000508      {-------------------------------------------------------------------------------------------------}
000509      BEGIN
000510  {$IFC fTraceUT}    LogCall;     {$ENDC}
000511          IF membersPlusHole <> (SELF.size + SELF.holeSize) THEN
000512              ResizeDynObject(SELF, SELF.dynStart + (membersPlusHole * SELF.MemberBytes));
000513      END;
000514
000515
000516  {$IFC WithUObject}
000517  {$S TKUTMain}
000518  {$ELSEC}
000519  {$S UTMain}
000520  {$ENDC}
000521      {-------------------------------------------------------------------------------------------------}
000522      PROCEDURE {TCollection.}ShiftColl(afterSrcIndex, afterDstIndex, howMany: INTEGER);
000523      {-------------------------------------------------------------------------------------------------}
```

```
000524          VAR memberBytes:    INTEGER;
000525              numBytes:       INTEGER;
000526              startAddr:      LONGINT;
000527              srcAddr:        LONGINT;
000528              dstAddr:        LONGINT;
000529      BEGIN
000530  {$IFC fTraceUT}    LogCall;     {$ENDC}
000531          IF (howMany > 0) AND (afterSrcIndex <> afterDstIndex) THEN
000532              BEGIN
000533              memberBytes := SELF.MemberBytes;
000534              numBytes := howMany * memberBytes;
000535
000536              startAddr := TpLONGINT(SELF)^ + SELF.dynStart;
000537              srcAddr := startAddr + afterSrcIndex * memberBytes;
000538              dstAddr := startAddr + afterDstIndex * memberBytes;
000539
000540              IF afterSrcIndex < afterDstIndex THEN
000541                  UTXferRight(Ptr(srcAddr), Ptr(dstAddr), numBytes)
000542              ELSE
000543                  UTXferLeft(Ptr(srcAddr), Ptr(dstAddr), numBytes);
000544              END;
000545      END;
000546
000547
000548  {$IFC WithUObject}
000549  {$S TKUTMain}
000550  {$ELSEC}
000551  {$S UTMain}
000552  {$ENDC}
000553      {-------------------------------------------------------------------------------------------------}
000554      PROCEDURE {TCollection.}StartEdit(withSlack: INTEGER);
000555      {-------------------------------------------------------------------------------------------------}
000556      BEGIN
000557  {$IFC fTraceUT}    LogCall;     {$ENDC}
000558          SELF.holeStd := withSlack;
000559      END;
000560
000561
000562  {$IFC WithUObject}
000563  {$S TKUTMain}
000564  {$ELSEC}
000565  {$S UTMain}
000566  {$ENDC}
000567      {-------------------------------------------------------------------------------------------------}
000568      PROCEDURE {TCollection.}StopEdit;
000569      {-------------------------------------------------------------------------------------------------}
000570      BEGIN
000571  {$IFC fTraceUT}    LogCall;     {$ENDC}
```

```
000572          IF SELF.holeStart < SELF.size THEN
000573              SELF.EditAt(SELF.size + 1, 0);
000574          SELF.ResizeColl(SELF.size);
000575          SELF.holeStd := 0;
000576          SELF.holeSize := 0;
000577      END;
000578
000579
000580  {$IFC WithUObject}
000581  {$S TKUTInit}
000582  {$ELSEC}
000583  {$S UTInit}
000584  {$ENDC}
000585  END;
000586
000587
000588  METHODS OF TUTArray;
000589
000590  {$IFC WithUObject}
000591  {$S TKUTMain}
000592  {$ELSEC}
000593  {$S UTMain}
000594  {$ENDC}
000595      {-------------------------------------------------------------------------------------------------}
000596      FUNCTION  {TArray.}CREATE(object: TUTObject; heap: THeap; initialSlack, bytesPerRecord: INTEGER): TUTArray;
000597      {-------------------------------------------------------------------------------------------------}
000598      BEGIN
000599  {$IFC fTraceUT}    LogCall;     {$ENDC}
000600          IF ODD(bytesPerRecord) THEN
000601              bytesPerRecord := bytesPerRecord + 1;
000602          IF object = NIL THEN
000603              object := NewDynObject(heap, THISCLASS, initialSlack * bytesPerRecord);
000604          SELF := TUTArray(TUTCollection.CREATE(object, heap, initialSlack));
000605          SELF.recordBytes := bytesPerRecord;
000606      END;
000607
000608
000609  {$IFC WithUObject}
000610  {$S TKUTMain}
000611  {$ELSEC}
000612  {$S UTMain}
000613  {$ENDC}
000614      {-------------------------------------------------------------------------------------------------}
000615      FUNCTION  {TArray.}MemberBytes: INTEGER;
000616      {-------------------------------------------------------------------------------------------------}
000617      BEGIN
000618  {$IFC fTraceUT}    LogCall;     {$ENDC}
000619          MemberBytes := SELF.recordBytes;
```

```
000620      END;
000621
000622
000623  {$IFC WithUObject}
000624  {$S TKUTMain}
000625  {$ELSEC}
000626  {$S UTMain}
000627  {$ENDC}
000628      {-------------------------------------------------------------------------------------------------}
000629      FUNCTION  {TArray.}At(i: LONGINT): Ptr;
000630      {-------------------------------------------------------------------------------------------------}
000631      BEGIN
000632  {$IFC fTraceUT}    LogCall;     {$ENDC}
000633         { At := Ptr(SELF.AddrMember(i));   but for speed...}
000634
000635          IF i > SELF.holeStart THEN
000636              i := i + SELF.holeSize;
000637
000638          At := Ptr(TpLONGINT(SELF)^ + SELF.dynStart + (SELF.recordBytes * (i - 1)));
000639      END;
000640
000641
000642  {$IFC WithUObject}
000643  {$S TKUTMain}
000644  {$ELSEC}
000645  {$S UTMain}
000646  {$ENDC}
000647      {-------------------------------------------------------------------------------------------------}
000648      PROCEDURE {TArray.}DelAll;
000649      {-------------------------------------------------------------------------------------------------}
000650      BEGIN
000651  {$IFC fTraceUT}    LogCall;     {$ENDC}
000652          SELF.EditAt(1, -SELF.size);
000653      END;
000654
000655
000656  {$IFC WithUObject}
000657  {$S TKUTMain}
000658  {$ELSEC}
000659  {$S UTMain}
000660  {$ENDC}
000661      {-------------------------------------------------------------------------------------------------}
000662      PROCEDURE {TArray.}DelAt(i: LONGINT);
000663      {-------------------------------------------------------------------------------------------------}
000664      BEGIN
000665  {$IFC fTraceUT}    LogCall;     {$ENDC}
000666          SELF.EditAt(i, -1);
000667      END;
```

```
000668
000669
000670  {$IFC WithUObject}
000671  {$S TKUTMain}
000672  {$ELSEC}
000673  {$S UTMain}
000674  {$ENDC}
000675      {-------------------------------------------------------------------------------------------------}
000676      PROCEDURE {TArray.}DelManyAt(i, howMany: LONGINT);
000677      {-------------------------------------------------------------------------------------------------}
000678      BEGIN
000679  {$IFC fTraceUT}    LogCall;     {$ENDC}
000680          SELF.EditAt(i, -howMany);
000681      END;
000682
000683
000684  {$IFC WithUObject}
000685  {$S TKUTMain}
000686  {$ELSEC}
000687  {$S UTMain}
000688  {$ENDC}
000689      {-------------------------------------------------------------------------------------------------}
000690      PROCEDURE {TArray.}PutAt(i: LONGINT; pRecord: Ptr);
000691      {-------------------------------------------------------------------------------------------------}
000692      BEGIN
000693  {$IFC fTraceUT}    LogCall;     {$ENDC}
000694          UTXferLeft(pRecord, Ptr(SELF.AddrMember(i)), SELF.recordBytes);
000695      END;
000696
000697
000698  {$IFC WithUObject}
000699  {$S TKUTMain}
000700  {$ELSEC}
000701  {$S UTMain}
000702  {$ENDC}
000703      {-------------------------------------------------------------------------------------------------}
000704      PROCEDURE {TArray.}InsAt(i: LONGINT; pRecord: Ptr);
000705      {-------------------------------------------------------------------------------------------------}
000706      BEGIN
000707  {$IFC fTraceUT}    LogCall;     {$ENDC}
000708          SELF.EditAt(i, 1);
000709          SELF.PutAt(i, pRecord);
000710      END;
000711
000712
000713  {$IFC WithUObject}
000714  {$S TKUTMain}
000715  {$ELSEC}
```

```
000716  {$S UTMain}
000717  {$ENDC}
000718      {-------------------------------------------------------------------------------------------------}
000719      PROCEDURE {TArray.}InsLast(pRecord: Ptr);
000720      {-------------------------------------------------------------------------------------------------}
000721      BEGIN
000722  {$IFC fTraceUT}    LogCall;     {$ENDC}
000723          {$IFC fTrce}BP(3);{$ENDC}
000724          SELF.InsAt(SELF.size + 1, pRecord);
000725          {$IFC fTrce}EP;{$ENDC}
000726      END;
000727
000728
000729  {$IFC WithUObject}
000730  {$S TKUTInit}
000731  {$ELSEC}
000732  {$S UTInit}
000733  {$ENDC}
000734  END;
000735
000736
000737  METHODS OF TUTString;
000738
000739  {$IFC WithUObject}
000740  {$S TKUTMain}
000741  {$ELSEC}
000742  {$S UTMain}
000743  {$ENDC}
000744      {-------------------------------------------------------------------------------------------------}
000745      FUNCTION  {TString.}CREATE(object: TUTObject; heap: THeap; initialSlack: INTEGER): TUTString;
000746      {-------------------------------------------------------------------------------------------------}
000747      BEGIN
000748  {$IFC fTraceUT}    LogCall;     {$ENDC}
000749          IF ODD(initialSlack) THEN
000750              initialSlack := initialSlack + 1;
000751          IF object = NIL THEN
000752              object := NewDynObject(heap, THISCLASS, initialSlack);
000753          SELF := TUTString(TUTCollection.CREATE(object, heap, initialSlack));
000754      END;
000755
000756
000757  {$IFC WithUObject}
000758  {$S TKUTMain}
000759  {$ELSEC}
000760  {$S UTMain}
000761  {$ENDC}
000762      {-------------------------------------------------------------------------------------------------}
000763      FUNCTION  {TString.}At(i: LONGINT): CHAR;
```

```
000764      {-------------------------------------------------------------------------------------------------}
000765      BEGIN
000766  {$IFC fTraceUT}    LogCall;     {$ENDC}
000767         {At := CHAR(TPByte(SELF.AddrMember(i))^);  but for speed...}
000768
000769          IF i > SELF.holeStart THEN
000770              i := i + SELF.holeSize;
000771          At := CHAR(TPByte(TpLONGINT(SELF)^ + SELF.dynStart + (i - 1))^);
000772      END;
000773
000774
000775  {$IFC WithUObject}
000776  {$S TKUTMain}
000777  {$ELSEC}
000778  {$S UTMain}
000779  {$ENDC}
000780      {-------------------------------------------------------------------------------------------------}
000781      PROCEDURE {TString.}DelAt(i: LONGINT);
000782      {-------------------------------------------------------------------------------------------------}
000783      BEGIN
000784  {$IFC fTraceUT}    LogCall;     {$ENDC}
000785          SELF.EditAt(i, -1);
000786      END;
000787
000788
000789  {$IFC WithUObject}
000790  {$S TKUTMain}
000791  {$ELSEC}
000792  {$S UTMain}
000793  {$ENDC}
000794      {-------------------------------------------------------------------------------------------------}
000795      PROCEDURE {TString.}DelAll;
000796      {-------------------------------------------------------------------------------------------------}
000797      BEGIN
000798  {$IFC fTraceUT}    LogCall;     {$ENDC}
000799          SELF.EditAt(1, -SELF.size);
000800      END;
000801
000802
000803  {$IFC WithUObject}
000804  {$S TKUTMain}
000805  {$ELSEC}
000806  {$S UTMain}
000807  {$ENDC}
000808      {-------------------------------------------------------------------------------------------------}
000809      PROCEDURE {TString.}DelManyAt(i, howMany: LONGINT);
000810      {-------------------------------------------------------------------------------------------------}
000811      BEGIN
```

```
000812  {$IFC fTraceUT}    LogCall;     {$ENDC}
000813          SELF.EditAt(i, -howMany);
000814      END;
000815
000816
000817  {$IFC WithUObject}
000818  {$S TKUTMain}
000819  {$ELSEC}
000820  {$S UTMain}
000821  {$ENDC}
000822      {-------------------------------------------------------------------------------------------------}
000823      PROCEDURE {TString.}InsAt(i: LONGINT; character: CHAR);
000824      {-------------------------------------------------------------------------------------------------}
000825          VAR pByte:      TPByte;
000826      BEGIN
000827  {$IFC fTraceUT}    LogCall;     {$ENDC}
000828          SELF.EditAt(i, 1);
000829
000830         {TPByte(SELF.AddrMember(i))^ := PByte(character);  but for speed...}
000831
000832          pByte := TPByte(TpLONGINT(SELF)^ + SELF.dynStart + (i - 1));
000833          pByte^ := TByte(character);
000834      END;
000835
000836
000837  {$IFC WithUObject}
000838  {$S TKUTMain}
000839  {$ELSEC}
000840  {$S UTMain}
000841  {$ENDC}
000842      {-------------------------------------------------------------------------------------------------}
000843      PROCEDURE {TString.}InsPAOCAt(i: LONGINT; pPackedArrayOfCharacter: Ptr; howMany: LONGINT);
000844      {-------------------------------------------------------------------------------------------------}
000845      BEGIN
000846  {$IFC fTraceUT}    LogCall;     {$ENDC}
000847          SELF.EditAt(i, howMany);
000848          UTXferLeft(pPackedArrayOfCharacter, Ptr(SELF.AddrMember(i)), howMany);
000849      END;
000850
000851
000852  {$IFC WithUObject}
000853  {$S TKUTMain}
000854  {$ELSEC}
000855  {$S UTMain}
000856  {$ENDC}
000857      {-------------------------------------------------------------------------------------------------}
000858      PROCEDURE {TString.}ToPAOCAt(i, howMany: LONGINT; pPackedArrayOfCharacter: Ptr);
000859      {-------------------------------------------------------------------------------------------------}
```

```
000860      BEGIN
000861  {$IFC fTraceUT}    LogCall;     {$ENDC}
000862          SELF.EditAt(i + howMany, 0);
000863          UTXferLeft(Ptr(SELF.AddrMember(i)), pPackedArrayOfCharacter, howMany);
000864      END;
000865
000866
000867  {$IFC WithUObject}
000868  {$S TKUTMain}
000869  {$ELSEC}
000870  {$S UTMain}
000871  {$ENDC}
000872      {-------------------------------------------------------------------------------------------------}
000873      FUNCTION  {TString.}MemberBytes: INTEGER;
000874      {-------------------------------------------------------------------------------------------------}
000875      BEGIN
000876  {$IFC fTraceUT}    LogCall;     {$ENDC}
000877          MemberBytes := 1;
000878      END;
000879
000880
000881  {$IFC WithUObject}
000882  {$S TKUTInit}
000883  {$ELSEC}
000884  {$S UTInit}
000885  {$ENDC}
000886  END;
000887
000888  {$ENDC}
000889
000890
000891  {$IFC fUniversalTextTrace}
000892  {$IFC WithUObject}
000893  {$S TKUTMain}
000894  {$ELSEC}
000895  {$S UTMain}
000896  {$ENDC}
000897  {-------------------------------------------------------------------------------------------------}
000898  PROCEDURE PrintRun;
000899  {-------------------------------------------------------------------------------------------------}
000900  VAR i:      INTEGER;
000901      tab:    TTabDescritor;
000902
000903  BEGIN
000904  {$IFC fTraceUT}    LogCall;     {$ENDC}
000905      {$IFC fTrce}BP(11);{$ENDC}
000906      {lpd, achad}
000907      WRITELN('the character Descriptor is ');
```

```
000908      FOR i := 1 TO activeStream.data.size DO
000909          WRITE(activeStream.data.At(i));
000910      WRITELN;
000911      WRITELN('   maxDataSize ', activeStream.maxDataSize);
000912
000913  {$H-}
000914      WITH activeStream.characterDescriptor DO
000915          BEGIN
000916          WRITELN('      font ', font);
000917          WRITELN('      face ');
000918          WRITELN('      Superscript ', Superscript);
000919          WRITELN('      keepOnSamePage ', keepOnSamePage);
000920          END;
000921      WRITELN('the paragraph Descriptor is ');
000922      WITH activeStream.paragraphDescriptor DO
000923          BEGIN
000924          WRITELN('      paraGraphStart ', paraGraphStart);
000925          WRITELN('      firstLineMargin ', firstLineMargin);
000926          WRITELN('      bodyMargin ', bodyMargin);
000927          WRITELN('      rightMargin ', rightMargin);
000928          WRITELN('      paraLeading ', paraLeading);
000929          WRITELN('      lineSpacing ', lineSpacing);
000930          WRITELN('      ', tabTable.size:2,' Tabs ');
000931          FOR i := 1 TO tabTable.size DO
000932              BEGIN
000933              tab := TTabDescritor(tabTable.At(i));
000934              WITH tab DO
000935                  BEGIN
000936                  WRITELN('           position ', position);
000937                  WRITE  ('           fillBetweenTabs ');
000938                  CASE fillBetweenTabs OF
000939                      tNoFill:        WRITELN('No fill');
000940                      tDotFill:       WRITELN('Dot fill');
000941                      tHyphenFill:    WRITELN('Hyphen fill');
000942                      tUnderLineFill: WRITELN('Underline fill');
000943                  END;{CASE}
000944                  WRITE  ('           tabType ');
000945                  CASE tabType OF
000946                      qLeftTab:       WRITELN('Left tab');
000947                      qCenterTab:     WRITELN('Center tab');
000948                      qRightTab:      WRITELN('Right tab');
000949                      qPeriodTab:     WRITELN('Decimal period tab');
000950                      qCommaTab:      WRITELN('Decimal comma tab');
000951                  END;{CASE}
000952                  END;
000953              END;
000954          WRITE  ('      paraType ');
000955          CASE paraType OF
```

```
000956              qLeftPara:      WRITELN('Left aligned');
000957              qCenterPara:    WRITELN('Centered');
000958              qRightPara:     WRITELN('Right aligned');
000959              qJustPara:      WRITELN('Justified');
000960          END;{CASE}
000961          WRITELN('      hasPicture ', hasPicture);
000962          END;
000963  {$H+}
000964      {$IFC fTrce}EP;{$ENDC}
000965  END;
000966  {$ENDC}
000967
000968
000969  {$IFC fUniversalTextTrace}
000970  {$IFC WithUObject}
000971  {$S TKUTMain}
000972  {$ELSEC}
000973  {$S UTMain}
000974  {$ENDC}
000975  {-----------------------------------------------------------------------------------------------------}
000976  PROCEDURE PrintLpd(theLpd: TALpd);
000977  {-----------------------------------------------------------------------------------------------------}
000978
000979      PROCEDURE WriteQuad(quad: TQuad);
000980      BEGIN
000981  {$IFC fTraceUT}    LogCall;     {$ENDC}
000982          CASE quad OF
000983              quadL: WRITELN('quadL');
000984              quadC: WRITELN('quadC');
000985              quadR: WRITELN('quadR');
000986              quadJ: WRITELN('quadJ');
000987          END;
000988      END;
000989
000990      PROCEDURE WriteTArpe(arpe: TArpe);
000991      VAR i:  INTEGER;
000992      BEGIN
000993  {$IFC fTraceUT}    LogCall;     {$ENDC}
000994          WITH arpe DO
000995              BEGIN
000996              WRITELN('    cb:             ', cb:1);
000997              WRITELN('    sy:             ', sy:1);
000998              WRITELN('    xLftFst:        ', xLftFst:1);
000999              WRITELN('    xLftBody:       ', xLftBody:1);
001000              WRITELN('    xRt:            ', xRt:1);
001001              WRITELN('    yLd:            ', yLd:1);
001002              WRITELN('    fill1:          ', fill1:1);
001003              WRITELN('    yLine:          ', yLine:1);
```

```
001004              WRITE  ('    quad:           ');
001005              WriteQuad(quad);
001006              WRITELN('    itbLim:         ', itbLim:1);
001007              WRITELN('    argtbd:');
001008              FOR i := 0 TO itbLim - 1 DO
001009                  {$R-}
001010                  WITH argtbd[i] DO
001011                      BEGIN
001012                      WRITELN('    [',i:0,']:');
001013                      WRITELN('        x:              ', x:1);
001014                  (*  WRITELN('        fill4:          ', fill4:1); *)
001015                      WRITE  ('        quad:           ');
001016                      WriteQuad(argtbd[i].quad);
001017                      WRITE  ('        tyfill:         ');
001018                      CASE tyfill OF
001019                          tyfillNil:  WRITELN('tyfillNil');
001020                          tyfillDots: WRITELN('tyfillDots');
001021                          tyfillHyph: WRITELN('tyfillHyph');
001022                          tyfillUL:   WRITELN('tyfillUL');
001023                      END;
001024
001025                      WRITELN('        chLdr:          ', chLdr:1);
001026                      END;
001027                  {$R+}
001028              END;
001029      END;
001030
001031
001032  BEGIN
001033  {$IFC fTraceUT}    LogCall;     {$ENDC}
001034      {$IFC fTrce}BP(11);{$ENDC}
001035
001036      WRITELN('——————— Lpd ———————');
001037
001038      WITH theLpd DO
001039          BEGIN
001040          WRITELN('ics:            ', ics:1);
001041          WRITELN('ilpd:           ', ilpd:1);
001042          WRITELN('fParSt:         ', fParSt);
001043          WRITELN('lp:             ', lp:1);
001044          WRITELN('lplim           ', lplim:1);
001045          WRITELN('lpson:          ', lpson:1);
001046          WRITELN('icsson:         ', icsson:1);
001047          WRITELN('tyset:');
001048          WITH tyset DO
001049              BEGIN
001050              WRITELN('   fRce:            ', tyset.fRce);
001051              WRITELN('   fParBnds:        ', tyset.fParBnds);
```

```
001052              WRITELN('   fRpe:            ', tyset.fRpe);
001053              END;
001054          WRITELN('lpFstPar:       ', lpFstPar:1);
001055          WRITELN('lpLimPar:       ', lpLimPar:1);
001056
001057          IF tyset.fRpe THEN
001058              IF rpe = NIL THEN
001059                  WRITELN('rpe:           NIL')
001060              ELSE
001061                  WITH rpe^ DO
001062                  BEGIN
001063                  WRITELN('rpe:');
001064                  WriteTArpe(rpe^);
001065                  END;
001066
001067          IF tyset.fRce THEN
001068              WITH arce DO
001069                  BEGIN
001070                  WRITELN('arce:');
001071                  WRITELN('    cb:             ', cb:1);
001072                  WRITELN('    fVan:           ', fVan:1);
001073                  WRITELN('    fBold:          ', fBold:1);
001074                  WRITELN('    fItalic:        ', fItalic:1);
001075                  WRITELN('    fUnderline:     ', fUnderline:1);
001076                  WRITELN('    fill4:          ', fill4:1);
001077                  WRITELN('    cbSuperscript:  ', cbSuperscript:1);
001078                  WRITELN('    ifnt:           ', ifnt:1);
001079                  WRITELN('    fKeep:          ', fKeep:1);
001080                  WRITELN('    fOutLine:       ', fOutLine:1);
001081                  WRITELN('    fShadow:        ', fShadow:1);
001082                  WRITELN('    fFillB:         ', fFillB:1);
001083                  WRITELN('    fFillC:         ', fFillC:1);
001084                  WRITELN('    fFillD:         ', fFillD:1);
001085                  WRITELN('    fFillE:         ', fFillE:1);
001086                  WRITELN('    fFillF:         ', fFillF:1);
001087                  END;
001088
001089          IF tyset.fRpe THEN
001090                  BEGIN
001091                  WRITELN('arpe:');
001092                  WriteTArpe(arpe);
001093                  END;
001094
001095          END;
001096
001097      WRITELN('————————————————');
001098      WRITELN;
001099      {$IFC fTrce}EP;{$ENDC}
```

```
001100  END;
001101  {$ENDC}
001102
001103
001104
001105
001106  {$IFC fUniversalTextTrace}
001107  {$IFC WithUObject}
001108  {$S TKUTMain}
001109  {$ELSEC}
001110  {$S UTMain}
001111  {$ENDC}
001112  {-----------------------------------------------------------------------------------------------------}
001113  PROCEDURE PrintAchad(achad: TAchad);
001114  {-----------------------------------------------------------------------------------------------------}
001115  VAR i:      INTEGER;
001116      size:   INTEGER;
001117
001118  BEGIN
001119  {$IFC fTraceUT}    LogCall;     {$ENDC}
001120      {$IFC fTrce}BP(11);{$ENDC}
001121      WITH achad DO
001122          BEGIN
001123          WRITELN('————— Achad ————————');
001124          WRITELN('ichFst:         ', ichFst:1);
001125          WRITELN('ichLim:         ', ichLim:1);
001126          IF rgch <> NIL THEN
001127              BEGIN
001128              size := ichlim - ichFst - 1;
001129              IF size >= 80 THEN
001130                  size := 79;
001131              FOR i := ichFst TO ichFst + size DO
001132                  {$R-}
001133                  IF rgch^[i] >= 32 THEN
001134                      WRITE(CHR(rgch^[i]))
001135                  ELSE
001136                      WRITE('<', rgch^[i]:0, '>');
001137                  {$R+}
001138              WRITELN;
001139              IF ichlim - ichFst >= 79 THEN
001140                  WRITELN('etc, etc...');
001141              END;
001142          WRITELN('————————————————————');
001143          WRITELN;
001144          END;
001145      {$IFC fTrce}EP;{$ENDC}
001146  END;
001147  {$ENDC}
```

```
001148
001149
001150
001151
001152  {$IFC fUniversalTextTrace}
001153  {$IFC WithUObject}
001154  {$S TKUTMain}
001155  {$ELSEC}
001156  {$S UTMain}
001157  {$ENDC}
001158  {-------------------------------------------------------------------------------------------------------}
001159  PROCEDURE PrintSecrets(achad: TAchad; theLpd: TALpd);
001160  {-------------------------------------------------------------------------------------------------------}
001161  BEGIN
001162  {$IFC fTraceUT}    LogCall;     {$ENDC}
001163      {$IFC fTrce}BP(11);{$ENDC}
001164      WRITELN('streamArrayIndex is ', secrets.streamArrayIndex);
001165      PrintLpd(theLpd);
001166      PrintAchad(achad);
001167      {$IFC fTrce}EP;{$ENDC}
001168  END;
001169  {$ENDC}
001170
001171
001172  {$IFC WithUObject}
001173  {$S TKUTWrite}
001174  {$ELSEC}
001175  {$S UTWrite}
001176  {$ENDC}
001177  {-------------------------------------------------------------------------------------------------------}
001178  PROCEDURE SeqLpdUTBB(Lpd: TLpd; var achad: Tachad);
001179  {-------------------------------------------------------------------------------------------------------}
001180  VAR howMany:    INTEGER;
001181      done:       BOOLEAN;
001182      index:      INTEGER;
001183      backUp:     INTEGER;
001184      newPara:    BOOLEAN;
001185  {$IFC WithUObject}
001186      newData:    TString;
001187  {$ELSEC}
001188      newData:    TUTString;
001189  {$ENDC}
001190  BEGIN
001191  {$IFC fTraceUT}    LogCall;     {$ENDC}
001192      {$IFC fTrce}BP(11);{$ENDC}
001193
001194                  {LSR: put the next assignment and the WITH before the debugging code because
001195                         PrintSecrets depends on it.}
```

```
001196      currentLpd := lpd;                                  { Remember the lpd for RunToStream }
001197
001198      WITH lpd^ DO                                        { Make shure the lpd is set up OK }
001199          BEGIN
001200          rpe := @arpe;
001201          rce := @arce;
001202          END;
001203
001204      {$IFC fUniversalTextTrace}
001205      IF fPrintSecrets THEN
001206          BEGIN
001207          WRITELN('———————————————————————————— SeqLpdUTBB ————————————————————————————');
001208          PrintSecrets(achad, currentLpd^);
001209          WRITELN('dataLp =       ', dataLp:0);
001210          WRITELN('dataIndex =    ', dataIndex:0);
001211          WRITELN('nOfSavedPara = ', nOfSavedPara:0);
001212          END;
001213      {$ENDC}
001214
001215      newPara := FALSE;                                   {Assume no new para}
001216                                                          { Compute if we have to back up }
001217      backUp := MIN(maxBacking, MAX(dataLp - lpd^.lpLim, 0));
001218
001219      {$IFC fUniversalTextTrace}
001220      IF fPrintSecrets THEN
001221          WRITELN('backUp =       ', backUp:0);
001222      {$ENDC}
001223
001224      IF backUp > 0 THEN
001225          BEGIN
001226          index := 1;
001227          done := FALSE;
001228          WHILE (NOT done) AND (index <= nOfSavedPara) DO
001229              WITH savedPara[index]^^, lpd^ DO
001230                  IF firstLp <= lpLim THEN
001231                      BEGIN
001232                      {$IFC fUniversalTextTrace}
001233                      IF fPrintSecrets THEN
001234                          WRITELN('Backing up... to saved paragraph #', index:0);
001235                      {$ENDC}
001236
001237                      lpLim := MAX(firstLp, lpLim);
001238                      theData := theText;
001239                      dataIndex := lpLim - firstLp;
001240                      arpe := theArpe;
001241                      IF dataIndex <> 0 THEN
001242                          fParSt := FALSE;
001243                      arce := theArce;
```

```
001244                      done := TRUE;
001245                      END
001246                  ELSE
001247                      index := index + 1;
001248
001249          IF NOT done THEN                                { This is FATAL !!!}
001250              BEGIN
001251              {$IFC fUniversalTextTrace}
001252              WRITELN('Fatal back up attempt in SeqLpdUTBB');
001253              PrintSecrets(achad, lpd^);
001254              {$ENDC}
001255
001256              HALT;   {Die rather than fuck up}
001257              END;
001258          END
001259      ELSE
001260          BEGIN
001261          IF activeStream.data.Size = 0 THEN              { Test if there is anything left in the buffer, }
001262              BEGIN
001263              newPara := TRUE;
001264              activeStream.ParagraphDescriptor.paraGraphStart := TRUE;
001265  {$IFC WithUObject}
001266              activeStream.ParagraphDescriptor.additionalChrInParagraph := 0;
001267  {$ENDC}
001268              activeStream.FillParagraph;                 { if not then try to get one more paragraph }
001269              activeStream.data.StopEdit;                 { Remove any holes from data }
001270
001271              {$IFC fUniversalTextTrace}
001272              IF fPrintSecrets THEN
001273                  BEGIN
001274                  WRITELN('FillRun returns:');
001275                  PrintRun;
001276                  END;
001277              {$ENDC}
001278
001279              dataIndex := 0;                             { Reset the index to the begining of the text}
001280
001281              WITH lpd^ DO                                { Pre-fill the lpd with standard data }
001282                  BEGIN
001283            {$H-} MoveRgch(@arpe, @arpeStd, arpeStd.cb); {$H+}
001284            {$H-} MoveRgch(@arce, @arceStd, arceStd.cb); {$H+}
001285                  dataLp := lpLim;
001286                  END;
001287
001288              activeStream.RunToStream;                   { Convert into stream format }
001289              END
001290          ELSE
001291              BEGIN
```

```
001292              {$IFC fUniversalTextTrace}
001293              IF fPrintSecrets THEN
001294                  WRITELN('Procede with the rest of the old run:');
001295              {$ENDC}
001296              dataIndex := Lpd^.lpLim - dataLp;
001297              IF dataIndex <> 0 THEN
001298                  lpd^.fParSt := FALSE;
001299              END;
001300
001301          theData := activeStream.data;
001302          END;
001303
001304
001305                                                  { Compute how many bytes to transfer this time }
001306      howMany := MIN(achad.ichLim - achad.ichFst, theData.size - dataIndex);
001307
001308      {$IFC fUniversalTextTrace}
001309      IF fPrintSecrets THEN
001310          BEGIN
001311          WRITELN('theData.size = ', theData.size:0);
001312          WRITELN('dataLp =       ', dataLp:0);
001313          WRITELN('dataIndex =    ', dataIndex:0);
001314          WRITELN('howMany =      ', howMany:0);
001315          WRITELN('newPara =      ', newPara:0);
001316          END;
001317      {$ENDC}
001318
001319      WITH lpd^ DO
001320          BEGIN
001321          lp := lpLim;
001322          lplim := lp + howMany;
001323          END;
001324
001325      WITH secrets.achad DO                       { Build our own achad just in case... }
001326          BEGIN
001327          rgch := POINTER(ORD4(theData.AddrMember(1)) + dataIndex);
001328          ichfst := 0;
001329          ichLim := howMany;
001330          END;
001331                                                  { Copy the achad }
001332                                                  { Check for NIL rgch.}
001333                                                  { If NIL then pass data else copy the data }
001334      IF achad.rgch = NIL THEN
001335          achad := secrets.achad
001336      ELSE
001337          BEGIN
001338          achad.ichlim := achad.ichFst + howMany;
001339          MoveAchad(achad, secrets.achad);
```

```
001340          END;
001341
001342      IF howMany = 0 THEN                                     { We are done, kill all saved stuff }
001343          FOR index := 1 TO nOfSavedPara DO
001344              BEGIN
001345              savedPara[index]^^.theText.Free;
001346              FreeH(HzFromH(TH(savedPara[index])), TH(savedPara[index]));
001347              END
001348      ELSE
001349          BEGIN
001350          IF newPara THEN                                   { New text in activeStream.data... }
001351              BEGIN
001352              done := FALSE;
001353              index := nOfSavedPara;
001354              WHILE (NOT done) AND (index > 0) DO             { Get ridd of old stuff }
001355                  WITH savedPara[index]^^ DO
001356                      IF (lpd^.lpLim - (firstLp + theText.size) ) >= maxBacking THEN {LSR}
001357                          BEGIN
001358                          theText.Free;
001359                          FreeH(HzFromH(TH(savedPara[index])), TH(savedPara[index]));
001360                          index := index - 1;
001361                          nOfSavedPara := nOfSavedPara - 1;
001362                          END
001363                      ELSE
001364                          done := true;
001365
001366                       {LSR: changed direction of following loop}
001367              FOR index := nOfSavedPara DOWNTO 1 DO               { Shift everything to free the first one }
001368                  savedPara[index + 1] := savedPara[index];
001369
001370              nOfSavedPara := nOfSavedPara + 1;
001371
001372                                                              { Make a new place to save old paragraphs }
001373              savedPara[1] := POINTER(HAllocate(THz(activeStream.Heap), SIZEOF(TSavedPara)));
001374              WITH savedPara[1]^^, lpd^ DO
001375                  BEGIN
001376                  firstLp := lp;
001377                  theArpe := arpe;
001378                  theArce := arce;
001379                  theText := activeStream.data;
001380                  END;
001381              END;
001382
001383          IF (dataIndex + howMany) >= activeStream.data.Size THEN
001384              BEGIN                                 { Make a fresh string, the old one is in savedPara[1] }
001385
001386                          {LSR: break up the assignment to activeStream.data to prevent dereferenced handles}
001387  {$IFC WithUObject}
```

```
001388              newData := TString.CREATE(NIL, activeStream.heap, activeStream.maxDataSize);
001389  {$ELSEC}
001390              newData := TUTString.CREATE(NIL, activeStream.heap, activeStream.maxDataSize);
001391  {$ENDC}
001392              activeStream.data := newData;
001393              activeStream.data.StartEdit(50);                {Allow holes}
001394              dataLp := lpd^.lpLim;                           {Make shure backUp will compute to zero}
001395              END;
001396          END;
001397
001398      {$IFC fUniversalTextTrace}
001399      IF fPrintSecrets THEN
001400          BEGIN
001401          PrintSecrets(achad, currentLpd^);
001402          WRITELN('dataLp =       ', dataLp:0);
001403          WRITELN('dataIndex =    ', dataIndex:0);
001404          WRITELN('nOfSavedPara = ', nOfSavedPara:0);
001405          WRITELN('————————————————————————————————————————————————————————————————————————————————');
001406          WRITELN;
001407          WRITELN;
001408          END;
001409      {$ENDC}
001410
001411      {$IFC fTrce}EP;{$ENDC}
001412  END;
001413
001414
001415  {$IFC WithUObject}
001416  METHODS OF TTKUnivText
001417  {$ELSEC}
001418  METHODS OF TUnivText
001419  {$ENDC}
001420
001421  {$IFC WithUObject}
001422  {$S TKUTMain}
001423  {$ELSEC}
001424  {$S UTMain}
001425  {$ENDC}
001426      {-------------------------------------------------------------------------------------------------}
001427  {$IFC WithUObject}
001428      FUNCTION  {TUnivText.}CREATE(object:  TObject;
001429                                   itsHeap: THeap;
001430                                   itsTString: TString;
001431                                   itsDataSize: INTEGER)
001432                                                   : TTKUnivText;
001433  {$ELSEC}
001434      FUNCTION  {TUnivText.}CREATE(object:  TUTObject;
001435                                   itsHeap: THeap;
```

```
001436                                  itsTString: TUTString;
001437                                  itsDataSize: INTEGER)
001438                                                       : TUnivText;
001439  {$ENDC}
001440      {-------------------------------------------------------------------------------------------------}
001441
001442  {$IFC WithUObject}
001443      VAR thisTabTable:   TArray;
001444  {$ELSEC}
001445      VAR thisTabTable:   TUTArray;
001446  {$ENDC}
001447          thisString:     ^Tsp;
001448
001449      BEGIN
001450  {$IFC fTraceUT}    LogCall;     {$ENDC}
001451          {$IFC fTrce}BP(11);{$ENDC}
001452          IF object = NIL THEN
001453  {$IFC WithUObject}
001454              object := NewObject(itsHeap, THISCLASS);
001455  {$ELSEC}
001456              object := NewUTObject(itsHeap, THISCLASS);
001457  {$ENDC}
001458
001459  {$IFC WithUObject}
001460          SELF := TTKUnivText(object);
001461  {$ELSEC}
001462          SELF := TUnivText(object);
001463  {$ENDC}
001464
001465          { Get the stream }
001466
001467          SELF.itsOurTString := itsTString = NIL;
001468          IF SELF.itsOurTString THEN
001469  {$IFC WithUObject}
001470              itsTString := TString.CREATE(NIL, itsHeap, itsDataSize);
001471  {$ELSEC}
001472              itsTString := TUTString.CREATE(NIL, itsHeap, itsDataSize);
001473  {$ENDC}
001474
001475          itsTString.StartEdit(50);                   {Allow holes}
001476          SELF.data := itsTString;
001477          SELF.maxDataSize := itsDataSize;
001478
001479  {$IFC WithUObject}
001480          thisTabTable := TArray.CREATE(NIL, itsHeap, 0, SIZEOF(TTabDescriptor));
001481  {$ELSEC}
001482          thisTabTable := TUTArray.CREATE(NIL, itsHeap, 0, SIZEOF(TTabDescriptor));
001483  {$ENDC}
```

```
001484          thisTabTable.StartEdit(5);
001485          SELF.paragraphDescriptor.tabTable := thisTabTable;
001486          {$IFC fTrce}EP;{$ENDC}
001487      END;
001488
001489
001490  {$IFC WithUObject}
001491  {$S TKUTMain}
001492  {$ELSEC}
001493  {$S UTMain}
001494  {$ENDC}
001495      {-------------------------------------------------------------------------------------------------}
001496      PROCEDURE {TUnivText.}Free;
001497      {-------------------------------------------------------------------------------------------------}
001498      BEGIN
001499  {$IFC fTraceUT}    LogCall;     {$ENDC}
001500          {$IFC fTrce}BP(11);{$ENDC}
001501          {If the dynamic array was not passed in then free it}
001502          IF SELF.itsOurTString THEN
001503              SELF.data.Free;
001504          SELF.paragraphDescriptor.tabTable.Free;
001505          SUPERSELF.Free;
001506          {$IFC fTrce}EP;{$ENDC}
001507      END;
001508
001509
001510      {$IFC fDebugMethods}
001511  {$IFC WithUObject}
001512  {$S TKUTMain}
001513  {$ELSEC}
001514  {$S UTMain}
001515  {$ENDC}
001516      {-------------------------------------------------------------------------------------------------}
001517      PROCEDURE {TUnivText.}Fields(PROCEDURE Field(nameAndType: S255));
001518      {-------------------------------------------------------------------------------------------------}
001519      BEGIN
001520  {$IFC fTraceUT}    LogCall;     {$ENDC}
001521          SUPERSELF.Fields(Field);
001522          Field('paraGraphStart: BOOLEAN');
001523  {$IFC WithUObject}
001524          Field('additionalChrInParagraph: INTEGER');
001525  {$ENDC}
001526          Field('firstLineMargin: INTEGER');
001527          Field('bodyMargin: INTEGER');
001528          Field('rightMargin: INTEGER');
001529          Field('paraLeading: INTEGER');
001530          Field('lineSpacing: BYTE');
001531  {$IFC WithUObject}
```

```
001532          Field('tabTable: TArray');
001533  {$ELSEC}
001534          Field('tabTable: TUTArray');
001535  {$ENDC}
001536          Field('paraType: BYTE');
001537          Field('hasPicture: BOOLEAN');
001538          Field('font: INTEGER');
001539          Field('face: BYTE');
001540          Field('superscript: BYTE');
001541          Field('keepOnSamePage: BOOLEAN');
001542          Field('maxDataSize: INTEGER');
001543  {$IFC WithUObject}
001544          Field('data: TString');
001545  {$ELSEC}
001546          Field('data: TUTString');
001547  {$ENDC}
001548          Field('itsOurTString: BOOLEAN');
001549          Field('');
001550      END;
001551      {$ENDC}
001552
001553
001554  {$IFC WithUObject}
001555  {$S TKUTWrite}
001556  {$ELSEC}
001557  {$S UTWrite}
001558  {$ENDC}
001559      {-------------------------------------------------------------------------------------------------}
001560      PROCEDURE {TUnivText.}RunToStream;
001561      {-------------------------------------------------------------------------------------------------}
001562      VAR i:          INTEGER;
001563          found:      BOOLEAN;
001564
001565      BEGIN
001566  {$IFC fTraceUT}    LogCall;     {$ENDC}
001567          {$IFC fTrce}BP(11);{$ENDC}
001568
001569          IF currentLpd^.tyset.fRce THEN
001570            { Convert the character descriptor }
001571              WITH SELF.characterDescriptor, currentLpd^.rce^ DO
001572                  BEGIN
001573
001574                { Set the face fields }
001575                  fbold       := bold IN face;
001576                  fitalic     := italic IN face;
001577                  funderline  := underline IN face;
001578                  foutline    := outline IN face;
001579                  fshadow     := shadow IN face;
```

```
001580
001581                  fvan := FALSE;      {No vanished runs}
001582
001583                { Because of the way lotus does fonts we have to convert the a real font to a lotus font }
001584                  found := FALSE;
001585                  i := 0;
001586                  WHILE (i <= ifntlst) AND NOT(found) DO
001587                      BEGIN
001588                      IF argfam[i] = font THEN
001589                          BEGIN
001590                          ifnt := i;
001591                          found := TRUE;
001592                          END;
001593                      i := i + 1;
001594                      END;
001595                  IF NOT found THEN ifnt := 0;
001596
001597                  cbSuperscript := superscript;
001598                  fKeep := keepOnSamePage;
001599                  END; { with }
001600
001601        { Convert the paragraph descriptor }
001602          WITH SELF.ParagraphDescriptor, currentLpd^, rpe^ DO
001603              BEGIN
001604              fParSt := paraGraphStart;
001605              IF paraGraphStart THEN
001606                  BEGIN
001607                          {LSR: added lpLim to the right side of each of the following assignments}
001608                  lpFstPar := lpLim;
001609  {$IFC WithUObject}
001610                  lpLimPar := lpLim + SELF.data.Size + additionalChrInParagraph;
001611  {$ELSEC}
001612                  lpLimPar := lpLim + SELF.data.Size;
001613  {$ENDC}
001614                  END;
001615
001616              IF tyset.fRpe THEN
001617                  BEGIN
001618                  xLftFst := firstLineMargin;
001619                  xLftBody := bodyMargin;
001620                  xRt := rightMargin;
001621                  yLd := paraLeading;
001622                  yLine := lineSpacing;
001623
001624                  CASE paraType OF
001625                      qLeftPara:      quad := quadL;
001626                      qCenterPara:    quad := quadC;
001627                      qRightPara:     quad := quadR;
```

```
001628                      qJustPara:      quad := quadJ;
001629                      OTHERWISE       quad := quadL;
001630                  END;{CASE}
001631
001632                  itbLim := tabTable.Size - 1;
001633              {$H-} SELF.TabTableToArgTbd; {$H+}    { This invalidates WITH statement!! }
001634                  END;
001635              END;
001636
001637          {$IFC fTrce}EP;{$ENDC}
001638      END;
001639
001640
001641  {$IFC WithUObject}
001642  {$S TKUTMain}
001643  {$ELSEC}
001644  {$S UTMain}
001645  {$ENDC}
001646      {-------------------------------------------------------------------------------------------------}
001647      PROCEDURE {TUnivText.}StreamToRun;
001648      {-------------------------------------------------------------------------------------------------}
001649      VAR ifnt:       INTEGER;
001650      BEGIN
001651  {$IFC fTraceUT}    LogCall;     {$ENDC}
001652          {$IFC fTrce}BP(11);{$ENDC}
001653        { do the format stuff }
001654          IF secrets.lpd.tyset.frce THEN
001655              WITH SELF.characterDescriptor, secrets.lpd.rce^ DO
001656                  BEGIN
001657                  font := argfam[ifnt];
001658                  face := [];
001659                  IF fbold THEN
001660                      face := face + [bold];
001661                  IF fitalic THEN
001662                      face := face + [italic];
001663                  IF funderline THEN
001664                      face := face + [underline];
001665                  IF foutline THEN
001666                      face := face + [outline];
001667                  IF fshadow THEN
001668                      face := face + [shadow];
001669                  superscript := cbSuperscript;
001670                  keepOnSamePage := fKeep;
001671                  END;
001672
001673          IF secrets.lpd.tyset.frpe THEN
001674              BEGIN
001675              WITH SELF.paragraphDescriptor, secrets.lpd.rpe^ DO
```

```
001676                  BEGIN
001677                  paraGraphStart := secrets.lpd.fParSt;
001678                  firstLineMargin := xLftFst;
001679                  bodyMargin := xLftBody;
001680                  rightMargin := xRt;
001681                  paraLeading := yLd;
001682                  lineSpacing := yLine;
001683                  hasPicture := FALSE; {not yet implemented}
001684
001685                  CASE quad OF
001686                      quadL:      paraType := qLeftPara;
001687                      quadC:      paraType := qCenterPara;
001688                      quadR:      paraType := qRightPara;
001689                      quadJ:      paraType := qJustPara;
001690                      OTHERWISE   paraType := qLeftPara;
001691                  END;{CASE}
001692
001693                  IF itbLim < 0 THEN                  { Resize the tab table and move the data }
001694                      itbLim := -1;
001695                  END;
001696
001697              SELF.ArgTbdToTabTable;
001698              END;
001699          {$IFC fTrce}EP;{$ENDC}
001700      END;
001701
001702
001703  {$IFC WithUObject}
001704  {$S TKUTWrite}
001705  {$ELSEC}
001706  {$S UTWrite}
001707  {$ENDC}
001708      {-----------------------------------------------------------------------------------------------}
001709      PROCEDURE {TUnivText.}TabTableToArgTbd;
001710      {-----------------------------------------------------------------------------------------------}
001711      VAR i:          INTEGER;
001712          temp:       INTEGER;
001713          ptrToTab:   Ptr;
001714          tab:        TTabDescriptor;
001715
001716      BEGIN
001717  {$IFC fTraceUT}    LogCall;     {$ENDC}
001718          {$IFC fTrce}BP(11);{$ENDC}
001719          temp := MIN(SELF.paragraphDescriptor.tabTable.size, magicTabMax);
001720          FOR i:= 1 to temp  DO
001721              BEGIN
001722              tab := TTabDescriptor(SELF.paragraphDescriptor.tabTable.At(i));
001723              {R$-}
```

```
001724              WITH tab, currentLpd^.rpe^.argTbd[i-1] DO
001725                  BEGIN
001726                  x := position;
001727
001728                  CASE tabType OF
001729                      qLeftTab:
001730                          quad := quadL;
001731                      qCenterTab:
001732                          quad := quadC;
001733                      qRightTab:
001734                          quad := quadR;
001735                      qPeriodTab:
001736                          BEGIN
001737                          quad := quadJ;
001738                          fDecimalComma := FALSE;
001739                          END;
001740                      qCommaTab:
001741                          BEGIN
001742                          quad := quadJ;
001743                          fDecimalComma := TRUE;
001744                          END;
001745                      OTHERWISE
001746                          quad := quadL;
001747                  END;{CASE}
001748
001749                  CASE fillBetweenTabs OF
001750                      tNoFill:
001751                          BEGIN
001752                          tyFill := tyFillNil;
001753                          chLdr := ORD(' ');
001754                          END;
001755                      tDotFill:
001756                          BEGIN
001757                          tyFill := tyFillDots;
001758                          chLdr := ORD('.');
001759                          END;
001760                      tHyphenFill:
001761                          BEGIN
001762                          tyFill := tyFillHyph;
001763                          chLdr := ORD('-');
001764                          END;
001765                      tUnderLineFill:
001766                          BEGIN
001767                          tyFill := tyFillUL;
001768                          chLdr := ORD('_');
001769                          END;
001770                      OTHERWISE
001771                          BEGIN
```

```
001772                          tyFill := tyFillNil;
001773                          chLdr := ORD(' ');
001774                          END;
001775                  END;{CASE}
001776
001777                  END;
001778              {$R+}
001779              END;
001780          currentLpd^.rpe^.itbLim := temp - 1;
001781          {$IFC fTrce}EP;{$ENDC}
001782      END;
001783
001784
001785  {$IFC WithUObject}
001786  {$S TKUTMain}
001787  {$ELSEC}
001788  {$S UTMain}
001789  {$ENDC}
001790      {-------------------------------------------------------------------------------------------------}
001791      PROCEDURE {TUnivText.}ArgTbdToTabTable;
001792      {-------------------------------------------------------------------------------------------------}
001793      VAR i:          INTEGER;
001794          tab:        TTabDescriptor;
001795
001796      BEGIN
001797  {$IFC fTraceUT}    LogCall;     {$ENDC}
001798          {$IFC fTrce}BP(11);{$ENDC}
001799          { Size down the tab array for writing }
001800          SELF.paragraphDescriptor.tabTable.DelAll;
001801          FOR i := 0 to secrets.lpd.rpe^.itbLim - 1 DO
001802              BEGIN
001803              {$R-}
001804              WITH tab, secrets.lpd.rpe^.argTbd[i] DO
001805                  BEGIN
001806                  position := x;
001807
001808                  CASE quad OF
001809                      quadL:      tabType := qLeftTab;
001810                      quadC:      tabType := qCenterTab;
001811                      quadR:      tabType := qRightTab;
001812                      quadJ:      IF fDecimalComma THEN
001813                                      tabType := qCommaTab
001814                                  ELSE
001815                                      tabType := qPeriodTab;
001816                      OTHERWISE   tabType := qLeftTab;
001817                  END;{CASE}
001818
001819                  CASE tyFill OF
```

```
001820                      tyFillNil:  fillBetweentabs := tNoFill;
001821                      tyFillDots: fillBetweentabs := tDotFill;
001822                      tyFillHyph: fillBetweentabs := tHyphenFill;
001823                      tyFillUL:   fillBetweentabs := tUnderLineFill;
001824                      OTHERWISE   fillBetweentabs := tNoFill;
001825                  END;{CASE}
001826
001827                  {$IFC fUniversalTextTrace}
001828                  IF fPrintSecrets THEN
001829                      BEGIN
001830                      WRITELN('Tab #', i + 1:0, ', tabType =', ORD(tabType):0, ', quad =', ORD(quad):0);
001831                      END;
001832                  {$ENDC}
001833                  END;
001834              {$R+}
001835              SELF.paragraphDescriptor.tabTable.InsLast(@tab);
001836              END;
001837          {$IFC fTrce}EP;{$ENDC}
001838      END;
001839
001840  {$IFC WithUObject}
001841  {$S TKUTInit}
001842  {$ELSEC}
001843  {$S UTInit}
001844  {$ENDC}
001845  BEGIN
001846  {$IFC fTraceUT}    LogCall;     {$ENDC}
001847      {$IFC fUniversalTextTrace}
001848      fPrintSecrets := FALSE;
001849      {$ENDC}
001850  END;
001851
001852
001853  {$IFC WithUObject}
001854  METHODS OF TTKReadUnivText
001855  {$ELSEC}
001856  METHODS OF TReadUnivText
001857  {$ENDC}
001858
001859  {$IFC WithUObject}
001860  {$S TKUTMain}
001861  {$ELSEC}
001862  {$S UTMain}
001863  {$ENDC}
001864      {-------------------------------------------------------------------------------------------------}
001865  {$IFC WithUObject}
001866      FUNCTION  {TReadUnivText.}CREATE(object:  TObject;
001867                                       itsHeap: THeap;
```

```
001868                                      itsTString: TString;
001869                                      itsDataSize: INTEGER;
001870                                      LevelOfGranularity: TLevelOfGranularity)
001871                                                                  : TTKReadUnivText;
001872  {$ELSEC}
001873      FUNCTION  {TReadUnivText.}CREATE(object:  TUTObject;
001874                                      itsHeap: THeap;
001875                                      itsTString: TUTString;
001876                                      itsDataSize: INTEGER;
001877                                      LevelOfGranularity: TLevelOfGranularity)
001878                                                                  : TReadUnivText;
001879  {$ENDC}
001880      {-------------------------------------------------------------------------------------------------}
001881      VAR index:      TB;
001882  {$IFC WithUObject}
001883          thisList:   TString;
001884  {$ELSEC}
001885          thisList:   TUTString;
001886  {$ENDC}
001887      BEGIN
001888  {$IFC fTraceUT}    LogCall;     {$ENDC}
001889          {$IFC fTrce}BP(11);{$ENDC}
001890
001891        { Establish the level of granularity for reading }
001892          WITH secrets.lpd.tyset DO
001893              BEGIN
001894              frce := UTCharacters IN LevelOfGranularity;
001895              frpe := UTparagraphs IN LevelOfGranularity;
001896              fParBnds := FALSE;
001897              END;
001898
001899          GetCSScrap(index);
001900          IF index = 0 THEN
001901              SELF := NIL
001902          ELSE
001903              BEGIN
001904              secrets.streamArrayIndex := index;
001905              IF object = NIL THEN
001906  {$IFC WithUObject}
001907                  object := NewObject(itsHeap, THISCLASS);
001908  {$ELSEC}
001909                  object := NewUTObject(itsHeap, THISCLASS);
001910  {$ENDC}
001911
001912  {$IFC WithUObject}
001913              SELF := TTKReadUnivText(TTKUnivText.CREATE(object, itsHeap, itsTString, itsDataSize));
001914  {$ELSEC}
001915              SELF := TReadUnivText(TUnivText.CREATE(object, itsHeap, itsTString, itsDataSize));
```

```
001916  {$ENDC}
001917
001918  {$IFC WithUObject}
001919              thisList := TString.CREATE(NIL, itsHeap, itsDataSize);
001920  {$ELSEC}
001921              thisList := TUTString.CREATE(NIL, itsHeap, itsDataSize);
001922  {$ENDC}
001923              thisList.StartEdit(50);                   {Allow holes}
001924              SELF.buffer := thisList;
001925
001926              SELF.dataBeforeTab := TRUE;
001927              SELF.Restart;               { Set up for reading from the beginning}
001928              END;
001929          {$IFC fTrce}EP;{$ENDC}
001930      END;
001931
001932
001933      {$IFC fDebugMethods}
001934  {$IFC WithUObject}
001935  {$S TKUTMain}
001936  {$ELSEC}
001937  {$S UTMain}
001938  {$ENDC}
001939      {-------------------------------------------------------------------------------------------------}
001940      PROCEDURE {TReadUnivText.}Fields(PROCEDURE Field(nameAndType: S255));
001941      {-------------------------------------------------------------------------------------------------}
001942      BEGIN
001943  {$IFC fTraceUT}    LogCall;     {$ENDC}
001944          SUPERSELF.Fields(Field);
001945  {$IFC WithUObject}
001946          Field('buffer: TString');
001947  {$ELSEC}
001948          Field('buffer: TUTString');
001949  {$ENDC}
001950          Field('');
001951      END;
001952      {$ENDC}
001953
001954
001955  {$IFC WithUObject}
001956  {$S TKUTMain}
001957  {$ELSEC}
001958  {$S UTMain}
001959  {$ENDC}
001960      {-------------------------------------------------------------------------------------------------}
001961      PROCEDURE {TReadUnivText.}Free;
001962      {-------------------------------------------------------------------------------------------------}
001963      BEGIN
```

```
001964  {$IFC fTraceUT}    LogCall;     {$ENDC}
001965          {$IFC fTrce}BP(11);{$ENDC}
001966          SELF.buffer.Free;
001967          SUPERSELF.Free;
001968          {$IFC fTrce}EP;{$ENDC}
001969      END;
001970
001971
001972  {$IFC WithUObject}
001973  {$S TKUTMain}
001974  {$ELSEC}
001975  {$S UTMain}
001976  {$ENDC}
001977      {-------------------------------------------------------------------------------------------}
001978      PROCEDURE {TReadUnivText.}Restart;
001979      {-------------------------------------------------------------------------------------------}
001980      BEGIN
001981  {$IFC fTraceUT}    LogCall;     {$ENDC}
001982          {$IFC fTrce}BP(11);{$ENDC}
001983          { Set up the Achad for reading from the beginning}
001984          WITH secrets.achad DO
001985              BEGIN
001986              ichFst := 0;
001987              ichLim := SELF.data.size;
001988              END;
001989
001990          secrets.lpd.lpLim := 0;
001991
001992          SELF.columnCount := 0;
001993          {$IFC fTrce}EP;{$ENDC}
001994      END;
001995
001996
001997  {$IFC WithUObject}
001998  {$S TKUTMain}
001999  {$ELSEC}
002000  {$S UTMain}
002001  {$ENDC}
002002      {-------------------------------------------------------------------------------------------}
002003      PROCEDURE {TReadUnivText.}ScanTable(VAR rows, tabColumns, tabStopColumns: INTEGER);
002004      {-------------------------------------------------------------------------------------------}
002005      VAR
002006          fieldOverflow:      BOOLEAN;
002007          fieldTerminator:    CHAR;
002008          lastTerminator:     CHAR;
002009          tabType:            TTabTypes;
002010          columnsInThisRow:   INTEGER;
002011          dataBeforeTab:      BOOLEAN;
```

```
002012
002013      BEGIN
002014  {$IFC fTraceUT}    LogCall;     {$ENDC}
002015          rows := 0;
002016          tabColumns := 1;
002017          tabStopColumns := 0;
002018          columnsInThisRow := 1;          {There is at least one column}
002019          SELF.dataBeforeTab := TRUE;     {Make shure ReadField doesn't skip any fields}
002020          dataBeforeTab := FALSE;
002021
002022          SELF.Restart;
002023          WHILE SELF.ReadField(1, fieldOverflow, fieldTerminator, tabType) DO
002024              BEGIN
002025              IF columnsInThisRow = 1 THEN
002026                  BEGIN
002027                  IF SELF.data.size > 0 THEN
002028                      dataBeforeTab := TRUE;
002029                  IF tabStopColumns < SELF.paragraphDescriptor.tabTable.size THEN
002030                      tabStopColumns := SELF.paragraphDescriptor.tabTable.size;
002031                  END;
002032              lastTerminatior := fieldTerminator;
002033              IF fieldTerminator = CHR(chCr) THEN
002034                  BEGIN
002035                  rows := rows + 1;
002036                  columnsInThisRow := 1;
002037                  {Check the tab table here}
002038                  END
002039              ELSE
002040                  IF fieldTerminator = CHR(chTab) THEN
002041                      BEGIN
002042                      columnsInThisRow := columnsInThisRow + 1;
002043                      IF columnsInThisRow > tabColumns THEN
002044                          tabColumns := columnsInThisRow;
002045                      END;
002046              END;
002047
002048          SELF.Restart;
002049
002050          IF (NOT dataBeforeTab) AND (tabColumn > 0) THEN
002051              tabColumns := tabColumns - 1;
002052          SELF.dataBeforeTab := dataBeforeTab;
002053
002054          IF lastTerminatior <> CHR(chCr) THEN
002055              rows := rows + 1;
002056
002057          {$IFC fUniversalTextTrace}
002058          IF fPrintSecrets THEN
002059              BEGIN
```

```
002060              WRITELN('ScanTable:');
002061              WRITELN('   dataBeforeTab:      ', dataBeforeTab);
002062              WRITELN('   tabColumns:         ', tabColumns);
002063              WRITELN('   tabStopColumns:     ', tabStopColumns);
002064              WRITELN('   rows:               ', rows);
002065              END;
002066          {$ENDC}
002067
002068      END;
002069
002070
002071  {$IFC WithUObject}
002072  {$S TKUTMain}
002073  {$ELSEC}
002074  {$S UTMain}
002075  {$ENDC}
002076      {-------------------------------------------------------------------------------------------------}
002077      FUNCTION  {TReadUnivText.}ReadField(    maxFieldSize: INTEGER;
002078                                          VAR fieldOverflow: BOOLEAN;
002079                                          VAR fieldTerminator: CHAR;
002080                                          VAR tabType: TTabTypes)
002081                                                              : BOOLEAN;
002082      {-------------------------------------------------------------------------------------------------}
002083  {$IFC WithUObject}
002084      VAR data:               TString;
002085          buffer:             TString;
002086  {$ELSEC}
002087      VAR data:               TUTString;
002088          buffer:             TUTString;
002089  {$ENDC}
002090          i:                  INTEGER;
002091          terminatorFound:    BOOLEAN;
002092          result:             BOOLEAN;
002093          oldSize:            INTEGER;
002094          newSize:            INTEGER;
002095          columnNr:           INTEGER;
002096          tab:                TTabDescriptor;
002097          ch:                 CHAR;
002098
002099          PROCEDURE ReadBuffer;
002100          BEGIN
002101              SELF.data := buffer;
002102              SELF.ReadRun;
002103              SELF.data := data;
002104          END;
002105
002106      BEGIN
002107  {$IFC fTraceUT}    LogCall;     {$ENDC}
```

```
002108          {$IFC fTrce}BP(11);{$ENDC}
002109
002110
002111          REPEAT
002112              buffer := SELF.buffer;
002113              data := SELF.data;
002114
002115              IF buffer.Size = 0 THEN                         { If there is no data then get some }
002116                  ReadBuffer;
002117
002118              fieldTerminator := CHR(0);
002119              fieldOverflow := FALSE;
002120
002121              data.DelAll;
002122              terminatorFound := FALSE;
002123              IF buffer.Size > 0 THEN                         { If there is still text to paste }
002124                  BEGIN
002125
002126                  tabType := qLeftTab;                       { Default tab type }
002127                  IF SELF.columnCount > 0 THEN
002128                      IF SELF.paragraphDescriptor.tabTable.size >= SELF.columnCount THEN
002129                          tabType := TTabDescriptor(
002130                                          SELF.paragraphDescriptor.tabTable.At(SELF.columnCount)
002131                                                  ).tabType;
002132                  SELF.columnCount := SELF.columnCount + 1;
002133                  columnNr := SELF.columnCount;
002134                  result := TRUE;
002135                  REPEAT
002136                      i := 0;
002137                      WHILE (i < buffer.Size) AND (NOT terminatorFound) DO
002138                          BEGIN
002139                          i := i + 1;
002140                          ch := buffer.At(i);
002141                          IF (ch = CHR(chTab)) OR (ch = CHR(chCr)) THEN
002142                              BEGIN
002143                              terminatorFound := TRUE;
002144                              fieldTerminator := ch;
002145                              IF fieldTerminator = CHR(chCr) THEN
002146                                  SELF.columnCount := 0;
002147                              END;
002148                          END;
002149
002150                      oldSize := data.Size;
002151                      newSize := oldSize + i;
002152                      IF terminatorFound THEN                 { Hide the terminating character, if any }
002153                          newSize := newSize - 1;
002154
002155                      IF newSize > maxFieldSize THEN
```

```
002156                          BEGIN
002157                          newSize := maxFieldSize;
002158                          fieldOverflow := TRUE;
002159                          END;
002160
002161                      IF newSize > oldSize THEN
002162                          data.InsManyAt(1 + data.size, buffer, 1, newSize - oldSize);
002163
002164                      buffer.DelManyAt(1, i);
002165
002166                      IF (NOT terminatorFound) AND (buffer.Size = 0) THEN
002167                          ReadBuffer;
002168                  UNTIL terminatorFound OR (buffer.Size = 0);
002169
002170                  {$IFC fUniversalTextTrace}
002171                  IF fPrintSecrets THEN
002172                      BEGIN
002173                      WRITELN('Buffer size is ',buffer.Size:1, ' data size is ',data.size:1);
002174                      FOR i := 1 to data.size DO
002175                          WRITE(data.At(i));
002176                      IF fieldTerminator = CHR(chTab) THEN
002177                          WRITE('<Tab>')
002178                      ELSE
002179                      IF fieldTerminator = CHR(chCr) THEN
002180                          WRITE('<Cr>')
002181                      ELSE
002182                          WRITE('<End of paste>');
002183                      WRITELN;
002184                      WRITELN('FieldOverflow is ', fieldOverflow);
002185                      END;
002186                  {$ENDC}
002187                  END
002188              ELSE
002189                  result := FALSE;
002190          UNTIL (NOT result) OR (columnNr > 1) OR SELF.dataBeforeTab;
002191          ReadField := result;
002192
002193          {$IFC fTrce}EP;{$ENDC}
002194      END;
002195
002196
002197  {$IFC WithUObject}
002198  {$S TKUTMain}
002199  {$ELSEC}
002200  {$S UTMain}
002201  {$ENDC}
002202      {-------------------------------------------------------------------------------------------}
002203      FUNCTION  {TReadUnivText.}ReadLine(    maxLineSize: INTEGER;
```

```
002204                                      VAR lineOverflow: BOOLEAN;
002205                                      VAR lineTerminator: CHAR)
002206                                                              : BOOLEAN;
002207      {-------------------------------------------------------------------------------------------------}
002208  {$IFC WithUObject}
002209      VAR data:               TString;
002210          buffer:             TString;
002211  {$ELSEC}
002212      VAR data:               TUTString;
002213          buffer:             TUTString;
002214  {$ENDC}
002215          i:                  INTEGER;
002216          terminatorFound:    BOOLEAN;
002217          oldSize:            INTEGER;
002218          newSize:            INTEGER;
002219          ch:                 CHAR;
002220
002221          PROCEDURE ReadBuffer;
002222          BEGIN
002223  {$IFC fTraceUT}    LogCall;     {$ENDC}
002224              SELF.data := buffer;
002225              SELF.ReadRun;
002226              SELF.data := data;
002227          END;
002228
002229      BEGIN
002230  {$IFC fTraceUT}    LogCall;     {$ENDC}
002231          {$IFC fTrce}BP(11);{$ENDC}
002232          buffer := SELF.buffer;
002233          data := SELF.data;
002234
002235          IF buffer.Size = 0 THEN                         { If there is no data then get some }
002236              ReadBuffer;
002237
002238          lineTerminator := CHR(0);
002239          lineOverflow := FALSE;
002240
002241          data.DelAll;
002242          terminatorFound := FALSE;
002243          IF buffer.Size > 0 THEN                         { If there is still text to paste }
002244              BEGIN
002245              ReadLine := TRUE;
002246              REPEAT
002247                  i := 0;
002248                  WHILE (i < buffer.size) AND (NOT terminatorFound) DO
002249                      BEGIN
002250                      i := i + 1;
002251                      ch := buffer.At(i);
```

```
002252                  IF ch = CHR(chCr) THEN
002253                      BEGIN
002254                      terminatorFound := TRUE;
002255                      lineTerminator := ch;
002256                      END;
002257                  END;
002258
002259              oldSize := data.Size;
002260              newSize := oldSize + i;
002261              IF terminatorFound THEN                    { Hide the terminating character, if any }
002262                  newSize := newSize - 1;
002263
002264              IF newSize > maxLineSize THEN
002265                  BEGIN
002266                  newSize := maxLineSize;
002267                  lineOverflow := TRUE;
002268                  END;
002269
002270              IF newSize > oldSize THEN
002271                  data.InsManyAt(1 + data.size, buffer, 1, newSize - oldSize);
002272
002273              buffer.DelManyAt(1, i);
002274
002275              IF (NOT terminatorFound) AND (buffer.Size = 0) THEN
002276                  ReadBuffer;
002277          UNTIL terminatorFound OR (buffer.Size = 0);
002278
002279          {$IFC fUniversalTextTrace}
002280          IF fPrintSecrets THEN
002281              BEGIN
002282              WRITELN('Buffer size is ',buffer.Size:1, ' data size is ',data.size:1);
002283              FOR i := 1 to data.size DO
002284                  WRITE(data.At(i));
002285              IF lineTerminator = CHR(chCr) THEN
002286                  WRITE('<Cr>')
002287              ELSE
002288                  WRITE('<End of paste>');
002289              WRITELN;
002290              WRITELN('LineOverflow is ', lineOverflow);
002291              END;
002292          {$ENDC}
002293          END
002294      ELSE
002295          ReadLine := FALSE;
002296
002297      {$IFC fTrce}EP;{$ENDC}
002298  END;
002299
```

```
002300
002301  {$IFC WithUObject}
002302  {$S TKUTMain}
002303  {$ELSEC}
002304  {$S UTMain}
002305  {$ENDC}
002306      {-------------------------------------------------------------------------------------------------}
002307      PROCEDURE {TReadUnivText.}ReadRun;
002308      {-------------------------------------------------------------------------------------------------}
002309      VAR error:  INTEGER;
002310          size:   LONGINT;
002311
002312      BEGIN
002313  {$IFC fTraceUT}    LogCall;     {$ENDC}
002314          {$IFC fTrce}BP(11);{$ENDC}
002315          BindUTDSeg(error);
002316
002317        { Size up the tab and data arrays to take the next run }
002318          SELF.data.DelAll;
002319          SELF.data.EditAt(1, SELF.maxDataSize);
002320
002321        { Set the achad to receive the next run }
002322          WITH secrets.achad DO
002323              BEGIN
002324              rgch := POINTER(SELF.data.AddrMember(1));
002325              ichFst := 0;
002326              ichLim := SELF.maxDataSize;
002327              END;
002328
002329          WITH secrets DO
002330              REPEAT
002331                { Get the next run }
002332                  IF lpd.lplim = 0 THEN
002333                      SetLpd(@Lpd, streamArrayIndex, 0, lpd.Tyset, achad)
002334                  ELSE
002335                      Seqlpd(@lpd, achad);
002336              UNTIL (NOT lpd.rce^.fvan) OR (achad.ichFst = achad.ichLim);
002337
002338          {$IFC fUniversalTextTrace}
002339          IF fPrintSecrets THEN
002340              PrintSecrets(secrets.achad, secrets.lpd);
002341          {$ENDC}
002342
002343        { Convert to Run }
002344          SELF.StreamToRun;
002345          WITH secrets.lpd DO
002346              BEGIN
002347              IF tyset.fRpe THEN
```

```
002348                  size := lpLimPar - lp {LSR: changed lpFstPar to lp}
002349              ELSE
002350                  size := lpLim - lp;
002351
002352              IF size > (lpLim-lp) THEN
002353                  size := lpLim - lp;
002354
002355              lpLim := lp + size;
002356              END;
002357          IF size < SELF.data.size THEN
002358              SELF.data.DelManyAt(size + 1, SELF.data.size - size);
002359
002360          UnBindUTDSeg(error);
002361          {$IFC fTrce}EP;{$ENDC}
002362      END;
002363
002364
002365  {$IFC WithUObject}
002366  {$S TKUTMain}
002367  {$ELSEC}
002368  {$S UTMain}
002369  {$ENDC}
002370      {-------------------------------------------------------------------------------------------------}
002371      FUNCTION  {TReadUnivText.}GetParaPicture(heap: THeap) : PicHandle;
002372      {-------------------------------------------------------------------------------------------------}
002373      BEGIN
002374  {$IFC fTraceUT}    LogCall;     {$ENDC}
002375          {$IFC fTrce}BP(11);{$ENDC}
002376          GetParaPicture := NIL;
002377          {$IFC fTrce}EP;{$ENDC}
002378      END;
002379
002380  {$IFC WithUObject}
002381  {$S TKUTInit}
002382  {$ELSEC}
002383  {$S UTInit}
002384  {$ENDC}
002385  END;
002386
002387
002388  {$IFC WithUObject}
002389  METHODS OF TTKWriteUnivText
002390  {$ELSEC}
002391  METHODS OF TWriteUnivText
002392  {$ENDC}
002393
002394  {$IFC WithUObject}
002395  {$S TKUTWrite}
```

```
{$ELSEC}
{$S UTWrite}
{$ENDC}
    {-------------------------------------------------------------------------------------------------}
{$IFC WithUObject}
        FUNCTION  {TWriteUnivText.}CREATE(object: TObject;
                                          itsHeap: THeap;
                                          itsTString: TString;
                                          itsDataSize: INTEGER)
                                                        : TTKWriteUnivText;
{$ELSEC}
        FUNCTION  {TWriteUnivText.}CREATE(object: TUTObject;
                                          itsHeap: THeap;
                                          itsTString: TUTString;
                                          itsDataSize: INTEGER)
                                                        : TWriteUnivText;
{$ENDC}
    {-------------------------------------------------------------------------------------------------}
    VAR
        ptrToolKitUT:   TPtrToolKitUT;
        error:          INTEGER;
        index:          TB;

        {$IFC PasteTrace}
        dbgCh:          CHAR;
        {$ENDC}

    BEGIN
        {$IFC PasteTrace}
        WRITE('Do you want to debug (Y/N): ');
        READ(dbgCh);
        fPrintSecrets := dbgCh IN ['Y', 'y'];
        {$ENDC}

{$IFC fTraceUT}    LogCall;     {$ENDC}
        {$IFC fTrce}BP(11);{$ENDC}
        BindUTDseg(error);
        IF error <> 0 THEN
            ABCBreak('BindUTDseg Error',error);

        index := IcsCreate(tycsFld, SIZEOF(ToolKitUT), POINTER(ORD(itsHeap)));
        {$R-}
        ptrToolKitUT := POINTER( rghcs^[index]^ );
        {$R+}
        WITH secrets DO
            BEGIN
            streamArrayIndex := index;
            lpd.tyset.fRpe := TRUE;
```

```
002444              lpd.tyset.fRce := TRUE;
002445              END;
002446
002447          WITH  ptrToolKitUT^ DO
002448              BEGIN
002449              cspd.argproc[IProcSeqLpd] := @SeqLpdUTBB;
002450              cspd.argproc[IProcFreeIcs] := Pointer(procnil);
002451              cspd.argproc[IProcPxHcs] := Pointer(procnil);
002452              cspd.argproc[IProcFindLpFixed] := @FindLpFstPar;
002453              cspd.argproc[IProcFSelLpBounds] := @TrueStdSelLpBounds;
002454              END;
002455
002456          secrets.streamArrayIndex := index;
002457
002458          nOfSavedPara := 0;                      {Nothing in the backLogBuffer}
002459          dataLp := 0;                            {Starting lpd}
002460
002461          IF object = NIL THEN
002462  {$IFC WithUObject}
002463              object := NewObject(itsHeap, THISCLASS);
002464  {$ELSEC}
002465              object := NewUTObject(itsHeap, THISCLASS);
002466  {$ENDC}
002467
002468  {$IFC WithUObject}
002469          SELF := TTKWriteUnivText(TTKUnivText.CREATE(object, itsHeap, itsTString, itsDataSize));
002470  {$ELSEC}
002471          SELF := TWriteUnivText(TUnivText.CREATE(object, itsHeap, itsTString, itsDataSize));
002472  {$ENDC}
002473
002474      { Get a default UT character and paragraph descriptors }
002475          WITH secrets DO
002476              BEGIN
002477              lpd.rpe := @lpd.arpe;
002478        {$H-} moveRgch(@lpd.arpe, @arpeStd, arpeStd.cb); {$H+}
002479
002480              Lpd.rce := @lpd.arce;
002481        {$H-} moveRgch(pointer(ord(lpd.rce)), @arceStd, arceStd.cb); {$H+}
002482              END;
002483
002484          SELF.StreamToRun;
002485
002486          activeStream := SELF;
002487          SELF.data.DelAll;
002488
002489          StartGetScrap(error);
002490          IF error <> 0 THEN
002491              ABCBreak('StartGetScrap Error',error);
```

```
002492
002493          PutCsScrap(index, error);
002494          IF error <> 0 THEN
002495              ABCBreak('PutCsScrap Error',error);
002496
002497          freeics(index);
002498
002499          EndGetScrap(error);
002500          IF error <> 0 THEN
002501              ABCBreak('EndGetScrap Error',error);
002502
002503          UnbindUTDseg(error);
002504          IF error <> 0 THEN
002505              ABCBreak('UnbindUTDseg Error',error);
002506          {$IFC fTrce}EP;{$ENDC}
002507      END;
002508
002509
002510  {$IFC WithUObject}
002511  {$S TKUTWrite}
002512  {$ELSEC}
002513  {$S UTWrite}
002514  {$ENDC}
002515      {-----------------------------------------------------------------------------------------------}
002516      PROCEDURE {TWriteUnivText.}FillParagraph;
002517      {-----------------------------------------------------------------------------------------------}
002518      BEGIN
002519  {$IFC fTraceUT}    LogCall;     {$ENDC}
002520          {$IFC fTrce}BP(11);{$ENDC}
002521  {$IFC WithUObject}
002522          ABCBreak('Failed to reimplement TTKWriteUnivText.FillParagraph',0);
002523  {$ELSEC}
002524          ABCBreak('Failed to reimplement TWriteUnivText.FillParagraph',0);
002525  {$ENDC}
002526          {$IFC fTrce}EP;{$ENDC}
002527      END;
002528
002529  {$IFC WithUObject}
002530  {$S TKUTInit}
002531  {$ELSEC}
002532  {$S UTInit}
002533  {$ENDC}
002534  END;

End of File -- Lines: 2534 Characters: 79766
```

```
================================================================================================
FILE: "UFIXUTEXT.TEXT"
================================================================================================

000001  UNIT UFixUText;
000002
000003
000004  {This unit fixes a bug with UText, where pasting universal text containing a 14 Point or
000005      20 Pitch font would crash with a check range error, accessing the uvFont array.  The
000006      only access was made in TInsertionPoint.InsertText.  To fix the problem we subclass
000007      TInsertionPoint.InsertText, but install a pointer to the revised method in TInsertionPoint's
000008      method table.}
000009
000010
000011  {$SETC CalcNumbers := FALSE}        {IF TRUE, calculate level/method numbers, else use CONSTs}
000012  {$SETC Debug := FALSE}
000013
000014  INTERFACE
000015
000016  {$E ERRORS}
000017  {$E+}
000018
000019  USES
000020      {$U LIBPL/UCLASCAL}     UClascal,
000021      {$U UObject}            UObject,
000022      {$U QuickDraw}          QuickDraw,
000023      {$U UDraw}              UDraw,
000024      {$U UABC}               UABC,
000025      {$U UUnivText}          UTKUniversalText,
000026      {$U UText}              UText;
000027
000028  TYPE
000029      TFixInsertionPoint = SUBCLASS OF TInsertionPoint
000030
000031          FUNCTION  TFixInsertionPoint.CREATE: TFixInsertionPoint; ABSTRACT;
000032
000033          PROCEDURE TFixInsertionPoint.InsertText(text: TText; isParaSelection: BOOLEAN;
000034                                                  isWordSelection: BOOLEAN;
000035                                                  universalText: BOOLEAN); OVERRIDE;
000036          END;
000037
000038
000039  IMPLEMENTATION
000040
000041  {$IFC fSymOK AND Debug}
000042  {$D+}
000043  {$ELSEC}
```

```
000044  {$D-}
000045  {$ENDC}
000046
000047  {$IFC fDbgOK AND Debug}
000048  {$R+}
000049  {$ELSEC}
000050  {$R-}
000051  {$ENDC}
000052
000053  VAR uvFont:                 ARRAY[1..19] OF TFontRecord;
000054      cFixInsertionPoint:     TClass;
000055
000056
000057  {$S FixText1}       {Caller and HackMethodTable must be in the same segment}
000058  {$IFC CalcNumbers}
000059  PROCEDURE Caller;
000060      VAR ip:         TInsertionPoint;
000061          t:          TText;
000062  BEGIN
000063      ip.InsertText(t, TRUE, TRUE, TRUE);
000064  END;
000065  {$ENDC}
000066
000067
000068  PROCEDURE HackMethodTables;
000069  {$IFC CalcNumbers AND DEBUG}
000070      LABEL   1,100;
000071  {$ELSEC}
000072      {$IFC CalcNumbers}
000073      LABEL   100;
000074      {$ENDC}
000075      {$IFC Debug}
000076      LABEL   1;
000077      {$ENDC}
000078  {$ENDC}
000079
000080  {$IFC NOT CalcNumbers}
000081      CONST
000082          levNum      =   6;
000083          methNum     =   2;
000084  {$ENDC}
000085
000086      TYPE
000087          TMethodArray = ARRAY [1..256] OF LONGINT;
000088          TPMethodArray = ^TMethodArray;
000089
000090          TSliceTable = ARRAY [0..255] OF TPMethodArray;
000091          TPSliceTable = ^TSliceTable;
```

```
000092
000093      VAR myProc:         LONGINT;
000094          {$IFC CalcNumbers}
000095          pc:             TpInteger;
000096          wd:             INTEGER;
000097          levNum:         INTEGER;
000098          methNum:        INTEGER;
000099          {$ENDC}
000100          pSliceTable:    TpSliceTable;
000101
000102  BEGIN
000103  {$IFC Debug}
000104  1:  GOTO 1;
000105  {$ENDC}
000106
000107  {$IFC CalcNumbers}
000108      {Find out the method # & level # for TInsertionPoint.InsertText}
000109      pc := TpInteger(@Caller);
000110      WHILE ORD(pc) <= ORD(@HackMethodTables) DO
000111          BEGIN
000112          wd := pc^;
000113          pc := TpInteger(ORD(pc)+2);
000114          IF wd = $4E95 THEN {JSR (A5)}
000115              BEGIN
000116              wd := pc^;                  {get level/method # as an integer}
000117              levNum := wd DIV 256;       {these 2 statements only work for <128 levels}
000118              methNum := wd MOD 256;
000119  {$IFC Debug}
000120              WriteLn(levNum, methNum); {***}
000121  {$ENDC}
000122              GOTO 100;
000123              END;
000124          END;
000125      HALT;           {did not find the method call}
000126  100:
000127  {$ENDC}
000128
000129      pSliceTable := TpSliceTable(cFixInsertionPoint);
000130      myProc := pSliceTable^[levNum]^[methNum];
000131
000132              {The superclass pointers have not been installed yet, so need to use the arrays in UClascal.}
000133      pSliceTable := TpSliceTable(pSTables^[pClasses^[numClasses].superIndex]);
000134      pSliceTable^[levNum]^[methNum] := myProc;
000135  END;
000136
000137  METHODS OF TFixInsertionPoint;
000138
000139  {$S FixText2}
```

```
000140      PROCEDURE TFixInsertionPoint.InsertText(text: TText; isParaSelection: BOOLEAN; isWordSelection: BOOLEAN;
000141                                                                                      universalText: BOOLEAN);
000142      VAR s:              TListScanner;
000143          prevPara:       TEditPara;
000144          newPara:        TEditPara;
000145          aParagraph:     TEditPara;
000146          newLP:          INTEGER;
000147          textImage:      TTextImage;
000148          insertIt:       BOOLEAN;
000149          done:           BOOLEAN;
000150          newParaImage:   TParaImage;
000151          paraIndex:      LONGINT;
000152          delta:          INTEGER;
000153          numParas:       INTEGER;
000154          needSpRight:    BOOLEAN;
000155          {$IFC fUseUnivText}
000156          readUnivText:   TTKReadUnivText;
000157          univPara:       TEditPara;
000158          univFormat:     TParaFormat;
000159          {$ENDC}
000160
000161          PROCEDURE StartPaste;
000162          BEGIN
000163              {$IFC fTrace}BP(10);{$ENDC}
000164              IF universalText THEN
000165                  BEGIN
000166                  {$IFC fUseUnivText}
000167                  univFormat := TParaFormat.CREATE(NIL, SELF.Heap, SELF.textImage.text.styleSheet);
000168                  univPara := textImage.NewEditPara(0, prevPara.format);
000169                  readUnivText := TTKReadUnivText.CREATE(NIL, SELF.Heap, NIL, 512,
000170                                                         [UTCharacters, UTParagraphs]);
000171                  numParas := 0;
000172                  {$ENDC}
000173                  END
000174              ELSE
000175                  BEGIN
000176                  numParas := text.paragraphs.size;
000177                  s := text.paragraphs.Scanner;
000178                  END;
000179              {$IFC fTrace}EP;{$ENDC}
000180          END;
000181
000182          PROCEDURE EndPaste;
000183          BEGIN
000184              {$IFC fTrace}BP(10);{$ENDC}
000185              IF universalText THEN
000186                  BEGIN
000187                  {$IFC fUseUnivText}
```

```
000188                  univPara.Free;
000189                  readUnivText.Free;
000190                  {$ENDC}
000191                  END;
000192              {$IFC fTrace}EP;{$ENDC}
000193          END;
000194
000195          FUNCTION GetParagraph(VAR paragraph: TEditPara): BOOLEAN;
000196          VAR currPos:        INTEGER;
000197              done:           BOOLEAN;
000198              runSize:        INTEGER;
000199              wasSomeText:    BOOLEAN;
000200              ch:             CHAR;
000201              typeStyle:      TTypeStyle;
000202          BEGIN
000203              {$IFC fTrace}BP(10);{$ENDC}
000204              If universalText THEN
000205                  BEGIN
000206                  {$IFC fUseUnivText}
000207                  univPara.ReplPString(0, univPara.Size, NIL);
000208                  currPos := 0;
000209                  wasSomeText := FALSE;
000210                  done := FALSE;
000211                  REPEAT
000212                      readUnivText.ReadRun;
000213                      runSize := readUnivText.data.size;
000214                      IF runSize > 0 THEN
000215                          BEGIN
000216                          IF NOT wasSomeText THEN
000217                              BEGIN
000218                              WITH univFormat, readUnivText.paragraphDescriptor DO
000219                                  BEGIN
000220                                  firstIndent := firstLineMargin;
000221                                  leftIndent := bodyMargin;
000222                                  (* Can't use this because it's given as distance from left rather than
000223                                     indent from right and I don't know what value of right edge of paper is.
000224                                  rightIndent := rightMargin;
000225                                  *)
000226                                  spaceBelowPara := paraLeading;
000227                                  END;
000228                              univPara.format := univFormat;
000229                              END;
000230                          wasSomeText := TRUE;
000231                          ch := readUnivText.data.At(runSize);
000232                          IF ORD(ch) = ascReturn THEN
000233                              BEGIN
000234                              readUnivText.data.DelAt(runSize);
000235                              runSize := runSize - 1;
```

```
000236                          numParas := numParas + 1;
000237                          done := TRUE;
000238                          END;
000239                      univPara.ReplTString(currPos, 0, readUnivText.data, 0, runSize);
000240                      typeStyle.onFaces := readUnivText.characterDescriptor.face;
000241                      typeStyle.font.fontFamily := uvFont[readUnivText.characterDescriptor.font].fontFamily;
000242                      typeStyle.font.fontSize := uvFont[readUnivText.characterDescriptor.font].fontSize;
000243                      univPara.NewStyle(currPos, currPos+runSize, typeStyle);
000244                      currPos := currPos + runSize;
000245                      END
000246                  ELSE
000247                      BEGIN
000248                      IF wasSomeText THEN
000249                          numParas := numParas + 1;
000250                      done := TRUE;
000251                      END;
000252              UNTIL done;
000253              IF wasSomeText THEN
000254                  paragraph := univPara
000255              ELSE
000256                  paragraph := NIL;
000257              GetParagraph := wasSomeText;
000258              {$ELSEC}
000259              paragraph := NIL;
000260              GetParagraph := FALSE;
000261              {$ENDC}
000262              END
000263          ELSE
000264              GetParagraph := s.Scan(paragraph);
000265          {$IFC fTrace}EP;{$ENDC}
000266      END;
000267
000268      PROCEDURE InsText;
000269      BEGIN
000270          {$IFC fTrace}BP(10);{$ENDC}
000271          delta := 0;
000272          textImage := SELF.textImage;
000273          newLP := SELF.textRange.firstLP;
000274          newPara := SELF.textRange.firstPara;
000275          prevPara := newPara;
000276          insertIt := FALSE;
000277
000278          IF isWordSelection THEN
000279              BEGIN
000280              needSpRight := newPara.Qualifies(newLP);
000281              IF newPara.Qualifies(newLP-1) THEN
000282                  BEGIN
000283                  newPara.InsertOneChar(' ', newLP);
```

```
000284                  newLP := newLP + 1;
000285                  delta := 1;
000286                  END;
000287              END;
000288
000289  (*          {special case: if first paragraph in text is designated a whole paragraph (by isParaSelection) AND
000290               if the insertion point (SELF) is at the end of the paragraph then we want to make a new
000291               paragraph rather than append it to the current paragraph and consequently set the flag that
000292               was supposed to prevent the first paragraph from being inserted}
000293              IF isParaSelection AND (prevPara.size = newLP) THEN
000294                  BEGIN
000295                  newPara := textImage.NewEditPara(0, prevPara.format);
000296                  newLP := 0;
000297                  insertIt := TRUE;
000298                  END;
000299  *)
000300              done := FALSE;
000301              StartPaste;
000302              IF GetParagraph(aParagraph) THEN
000303                  BEGIN
000304                  delta := delta + aParagraph.size;
000305                  REPEAT
000306                  newPara.ReplPara(newLP, 0, aParagraph, 0, aParagraph.size);
000307                  newLP := newLP + aParagraph.size;
000308                  IF insertIt THEN
000309                      textImage.text.InsParaAfter(prevPara, newPara);
000310                  insertIt := TRUE;
000311                  prevPara := newPara;
000312                  IF GetParagraph(aParagraph) THEN
000313                      BEGIN
000314                      newPara := textImage.NewEditPara(prevPara.size-newLP,
000315                                                       TParaFormat(aParagraph.format.Clone(SELF.Heap)));
000316                      {For now, so we don't get garbage (if aParagraph later deleted), put cloned
000317                       format on to styleSheet list}
000318                      SELF.textImage.text.styleSheet.formats.InsLast(newPara.format);
000319                      newPara.StartEdit(newPara.GrowSize);
000320                      newPara.ReplPara(0, 0, prevPara, newLp, prevPara.size - newLp);
000321                      prevPara.ReplPString(newLp, prevPara.size-newLP, NIL);
000322                      prevPara.StopEdit;
000323                      newLP := 0;
000324                      END
000325                   ELSE
000326                      done := TRUE;
000327                  UNTIL done;
000328                  END;
000329
000330              IF isParaSelection THEN
000331                  BEGIN
```

```
000332                  newPara := textImage.NewEditPara(prevPara.size - newLP, prevPara.format);
000333                  newPara.StartEdit(newPara.GrowSize);
000334                  newPara.ReplPara(0, 0, prevPara, newLp, prevPara.size - newLp);
000335                  prevPara.ReplPString(newLp, prevPara.size - newLP, NIL);
000336                  prevPara.StopEdit;
000337                  textImage.text.InsParaAfter(prevPara, newPara);
000338                  newPara := TEditPara(textImage.text.paragraphs.At(SELF.textRange.firstIndex + numParas));
000339                  numParas := numParas+1;
000340                  newLP := 0;
000341                  END
000342              ELSE IF isWordSelection THEN
000343                  IF needSpRight THEN
000344                      BEGIN
000345                      newPara.InsertOneChar(' ', newLP);
000346                      newLP := newLP + 1;
000347                      delta := delta + 1;
000348                      END;
000349
000350              EndPaste;
000351              {$IFC fTrace}EP;{$ENDC}
000352          END;
000353
000354          PROCEDURE Adjust;
000355              PROCEDURE AddDelta(paraImage: TParaImage);
000356              BEGIN
000357                  paraImage.AdjustLineLPs(SELF.textRange.firstLP, delta);
000358              END;
000359          BEGIN
000360              {$IFC fTrace}BP(10);{$ENDC}
000361              SELF.textRange.firstPara.EachImage(AddDelta);
000362
000363              WITH SELF, textRange DO
000364                  BEGIN
000365                  firstPara := newPara;
000366                  lastPara := newPara;
000367                  firstLP := newLP;
000368                  lastLP := newLP;
000369                  firstIndex := firstIndex + numParas - 1;
000370                  lastIndex := firstIndex;
000371                  newestLP := newLP;
000372                  amTyping := FALSE;
000373                  END;
000374              {$IFC fTrace}EP;{$ENDC}
000375          END;
000376      BEGIN
000377          {$IFC fTrace}BP(11);{$ENDC}
000378          IF (text <> NIL) OR universalText THEN
000379              SELF.ChangeText(InsText, Adjust);
```

```
000380          {$IFC fTrace}EP;{$ENDC}
000381      END;
000382
000383  {$S FixText1}
000384  BEGIN
000385      cFixInsertionPoint := THISCLASS;
000386
000387      HackMethodTables;
000388
000389      uvFont[4].fontFamily := famModern;
000390      uvFont[5].fontFamily := famModern;
000391      uvFont[6].fontFamily := famModern;
000392      uvFont[7].fontFamily := famModern;
000393      uvFont[8].fontFamily := famModern;
000394      uvFont[9].fontFamily := famModern;
000395      uvFont[10].fontFamily := famClassic;
000396      uvFont[11].fontFamily := famClassic;
000397      uvFont[12].fontFamily := famClassic;
000398      uvFont[13].fontFamily := famClassic;
000399      uvFont[14].fontFamily := famClassic;
000400      uvFont[15].fontFamily := famModern;         {added}
000401      uvFont[16].fontFamily := famClassic;        {added}
000402      uvFont[19].fontFamily := famModern;         {added}
000403      uvFont[4].fontSize := 5;
000404      uvFont[5].fontSize := 7;
000405      uvFont[6].fontSize := 8;
000406      uvFont[7].fontSize := 2;
000407      uvFont[8].fontSize := 3;
000408      uvFont[9].fontSize := 4;
000409      uvFont[10].fontSize := 5;
000410      uvFont[11].fontSize := 7;
000411      uvFont[12].fontSize := 8;
000412      uvFont[13].fontSize := 3;
000413      uvFont[14].fontSize := 6;
000414      uvFont[15].fontSize := 6;                   {added}
000415      uvFont[16].fontSize := 6;                   {added}
000416      uvFont[19].fontSize := 1;                   {added}
000417  END;
000418
000419
000420  END.
000421

End of File -- Lines: 421 Characters: 13961

SUMMARY:

  Total number of files : 38
```

Total file lines : 47256
Total file characters : 1553575