Text Building Block

To: Users of the ToolKit's Text Building Block

From:

Date: 21 March 1984

Subject: How to use the Text Building Block (TK8E) (draft version 3)

==========

When reading this document it is suggested that you have a copy of the UText interface available for reference.

The Text Building Block allows users to specify an **LRect** in their view in which multiple paragraphs may be entered either from the keyboard or pasted from the clipboard. Within this box, text will word wrap based on the bounds of the box and optional indentation fields in the **TParaFormat** object. These fields default to zero, thus if the values are not changed, the text will fit tightly within the LRect specified. (Currently only left justified text is supported so there will be a ragged right edge.)

Application programmers must create a **TTextImage** for each separate group of text they wish to display. The **CREATE** method expects an **LRect** within its view. In addition, the **isGrowable** boolean parameter specifies whether or not the LRect can grow at the bottom if text that is displayed cannot fit in the specified LRect.

TextImages display **TText** objects. A **TText** object contains a list of **TParagraph** objects. Each **TParagraph** object contains the characters and typeStyle information for one paragraph. Associated with each paragraph is a **TParaImage** which is stored in a list of **TParaImages** in **TTextImage**. A **TParaImage** contains all of the display specific information about a particular paragraph such as where each line ends and the pixel height of the displayed paragraph. When a **TTextImage** is created, it must have an initial **TText** object. The **TText** object, in turn, must have at least one paragraph in its paragraph list. Also, each paragraph points to a **TParaFormat**, which specifies margins, tabs, paragraph spacing and other formatting characteristics.

TParaFormats are stored in a **TStyleSheet**, which is simply a list of **TParaFormats**. Since the list is an indexed list, one can store commonly used formats at known positions in the list for rapid retrieval when reformatting a particular paragraph. Typically, applications store a single styleSheet in a field of their application window. Each **TText** object, in addition to containing a list of paragraphs also points to a styleSheet. By storing the styleSheet in the window, each **TText** object can reference the same styleSheet.

The typical order of allocation is then: Create a **TParaFormat**, then create a **TStyleSheet** and install the paraFormat in its list. Set **TAppWindow.styleSheet** to the new styleSheet. Create a **paragraph**, passing the **paraFormat** just created. Create a **TText** object and install the **paragraph** just created in its paragraphs list and the set **styleSheet** field to the new **styleSheet**. Now create the **TTextImage**. The **text** object has a list of text images that are displaying it (usually just one), so the **textImage** just created must be installed into the text's **txtImgList**. Before displaying the **textImage** for the first time the user must call **textImage.tk** so that the textImage's **paraImage** list will be initialized properly. The following code segment illustrates this:

```
paraFormat := TParaFormat.CREATE(NIL, heap);
styleSheet := TStyleSheet.CREATE(NIL, heap);
styleSheet.formats.InsLast(paraFormat);
myWindow.styleSheet := styleSheet;

paragraph := TParagraph.CREATE(NIL, heap, 0, paraFormat);
text := TText.CREATE(NIL, heap, styleSheet);
text.paragraphs.InsLast(paragraph);
textImage := TTextImage.CREATE(NIL, heap, myView, myLRect, text, TRUE);
text.txtImgList.InsLast(textImage);
textImage.RecomputeImages(actionNone, FALSE);
```

The parameters to RecomputeImages tell the Building Block not to draw anything (**actionNone**; other possible values are **actionDraw** which would draw directly on the screen if view.OKToDrawIn returns true, and **actionInval** which invalidates the portion of the screen affected by the change. Since we are in an initialization procedure, BlankStationery, there is no place to draw yet so we use **actionNone**.) and to set any invalid flags to FALSE, indicating that the data structures are now valid.

To simplify the above, the Text Building Block provides some Initialization methods to generate default objects. This is accomplished by the following:

```
styleSheet := TStyleSheet.CREATE(NIL, heap);
styleSheet.InitDefault;
myWindow.styleSheet := styleSheet;
text := TText.CREATE(NIL, heap, styleSheet);
textImage := text.DfltTextImage(myView, myLRect, isGrowable);
myView.textImage := textImage;
```

TStyleSheet.InitDefault will create a default paraFormat and install it in its formats list. **TText.DfltTextImage** will install a single empty paragraph in its paragraphs list. It uses its own styleSheet to get the paraFormat for the paragraph. It also creates a TTextImage object and installs it into its txtImgList and calls textImage.RecomputeImages to set up the paraImage list. Finally it returns the created textImage.>>

Once a textImage has been created and installed in the application's view, it can be displayed by calling **textImage.Draw** from the view's Draw method. If the text is to be framed by a box, the view's Draw method should first set a local LRect variable, r, to the textImage's extentLRect, call **InsetLRect(r, -1, 0)** and then call **FrameLRect(r)**. The InsetLRect makes the box one pixel wider on each side. The reason for this is that text highlighting always highlights by an

extra pixel so that characters at the edges will be completely highlighted. Of course, one could always make the box even bigger if desired.

In order to perform editing on a textImage, a **TTextSelection** must be established. This is done automatically by the Text Buliding Block when **TTextImage.MousePress** is called. Within the application view's **MousePress** routine will be tests to see if the mouseLPt falls within a particular textImage's extentLRect. **FoundTextImage.MousePress** is then called with the same mouseLPt as the parameter. TextImage.MousePress calls **panel.BeginSelection**, which automatically unhighlights the current selection. MousePress then determines which character the mouse was over and creates and blinks an insertion point at that character. (Note: **TInsertionPoint** is a subclass of **TTextSelection**.) If the mouse was somewhere after the last character then the insertion point appears immediately after the last character. Here is a simple example:

```
PROCEDURE TMyView.MousePress(mouseLPt: LPoint);
BEGIN
    IF LPtInLRect(mouseLPt, SELF.firstTxtImg.extentLRect) THEN
        SELF.firstTxtImg.MousePress(mouseLPt)
    ELSE IF LPtInLRect(mouseLPt, SELF.secondTxtImg.extentLRect) THEN
        SELF.secondTxtImg.MousePress(mouseLPt)
    ELSE
        {Do my own thing}
END;
```

From an application standpoint, this is all the programmer needs to do with regard to simple text images. The text building block handles most everything else via the textSelection.

If the user clicks and then drags the mouse, characters will be selected as they are encountered. Double and triple click are also handled automatically by the buliding block. When one or more characters are selected, the user may cut or copy the text to the clipboard and paste the text elsewhere. TypeStyle changes are also handled. When the clear command is executed, all the text in the text image is cleared, leaving just an insertion point.

Undo is supported for typing, style changes, cut, copy, paste, and clear. Undo typing will even work if the style is changed while typing. For example, if the user sets an insertion point, types some characters, selects underline, types some more characters, selects plain, types more and then selects "Undo Typing", all the characters entered since the insertion point was set are deleted.

More Sophisticated Use of the Text Building Block

Chained Text Images

The Text Building Block also supports the ability to have text flow from one textImage to another. This is accomplished via fields in **TTextImage** that must be set and maintained by the application. These fields are **nextTxtImg**,

prevTxtImg, **headTxtImg**, and **tailTxtImg**. The next and previous fields default to NIL and the head and tail fields default to SELF. The text field in these "linked" textImages all point to the same TText object. The paraImage list refers only to the images within the particular text image. Note that paragraphs may now be split between two (or more) textImages. Thus more than one paraImage may reference the same paragraph. The fields **startLP** and **endLP** in **TParaImage** indicate which characters in the paragraph it is displaying. (LP stands for logical position, where the first character in a paragraph is LP 0.)

When multiple textImages are linked together to display the same Text object, the application view's Draw method need only call **textImage.Draw** on the first textImage in the chain, the "head" textImage. TextImage.Draw calls Draw on each textImage in the chain. Here is a sample of BlankStationery code that would create three columns of text side by side:

```
SetLRect(r, 10, 10, 210, 310);

textImage := TTextImage.CREATE(NIL, heap, myView, r, text, FALSE);
text.txtImgList.InsLast(textImage);
myView.column1 := textImage;

OffsetLRect(r, 210, 0);
textImage := TTextImage.CREATE(NIL, heap, myView, r, text, FALSE);
myView.column2 := textImage;

OffsetLRect(r, 210, 0);
textImage := TTextImage.CREATE(NIL, heap, myView, r, text, FALSE);
myView.column3 := textImage;

WITH myView DO
    BEGIN
    column1.nextTxtImg := column2;
    column1.tailTxtImg := column3;
    column2.prevTxtImg := column1;
    column2.nextTxtImg := column3;
    column2.headTxtImg := column1;
    column2.tailTxtImg := column3;
    column3.prevTxtImg := column2;
    column3.headTxtImg := column1;
    END;

myView.column1.RecomputeImages(actionNone, FALSE);
```

Note a few important points in this example. We assume the TText object has already been created. In **TTextImage.CREATE** we passed FALSE for **isGrowable**. If we passed TRUE, text would never flow to the next box; instead the box would just get longer. Also, we only installed the first textImage into text.txtImgList. **This is very important.** Only the head textImage of a chained textImage list is installed in text.txtImgList. In the WITH statement, we took advantage of the default values of nextTxtImg, prevTxtImg, and so forth so we didn't have to set, for example, column1.prevTxtImg.

Multiple panels

A simple example of text in multiple panels is LisaCalc where the text in the selected cell is also shown in the top panel's "wide view". The Text Building

Block supports this notion. It is here that the txtImgList in the TText object is used. Where as the chained text images ability allows the same text object to flow from textImage to textImage, the txtImgList allows the same portion of the text to be visible in multiple panels at once. When a textImage is created that displays the same text object as in another panel, the textImage is simply installed into the text's txtImgList. Here is another code segement from BlankStationery:

```
panel := TPanel.CREATE(NIL, heap, SELF, ... );
myTopView := TMyTopView.CREATE(NIL, heap, panel, ... );
SetLRect(r, 10, 10, 210, 310);
textImage := TTextImage.CREATE(NIL, heap, myTopView, r, text, TRUE);
text.txtImgList.InsLast(textImage);
myTopView.textImage := textImage;
textImage.RecomputeImages(actionNone, FALSE);

panel := panel.Divide(v, ... );
myBottomView := TMyBottomView.CREATE(NIL, heap, panel, ... );
SetLRect(r, 50, 10, 600, 150);
textImage := TTextImage.CREATE(NIL, heap, myBottomView, r, text, FALSE);
text.txtImgList.InsLast(textImage);
myBottomView.textImage := textImage;
textImage.RecomputeImages(actionNone, FALSE);
```

Again, we assume the TText object has already been created. Note this time that we install both textImages into text.txtImgList. Also, notice that the two textImages have completely different sized extentLRects and one is growable and the other is not. This is perfectly acceptable. The text will be be formatted differently in each panel. All changes to text in one panel will be reflected in the text displayed in the other panel. This includes highlighting and insertion point blinking.

To reiterate a previous point, an important assumption that the Text Building Block makes is that each textImage in text.txtImgList is in a different panel. Applications must adhere to this rule.

```
1    1 --   UNIT UText;
1    2 --   {$SETC IsIntrinsic := TRUE }
1    3 --
1    4 --   {$IFC IsIntrinsic}
1    5 --   !NTRINSIC;
1    6 --   {$ENDC}
1    7 --
1    8 --
1    9 --   {Multiple Paragraph Building Block for the Tool Kit}
1   10 --
1   11 --   {changed 04/25/84 1437  Added TTextImage.TxtImgForClipBoard method}
1   12 --   {changed 04/18/84 1652  Added firstLinePixel, useFirstPixel fields to TTextImage}
1   13 --   {changed 04/16/84 1135  Added styleSheet field to TParaFormat}
1   14 --   {changed 04/13/84 0209  Added TTextImage.NewEditPara}
1   15 --   {changed 04/12/84 2344  Changed parameter list of TParagraph.UpdateRuns}
1   16 --   {changed 04/10/84 1400  Changed TEditPara.images field back to a TList}
1   17 --
1   18 --   INTERFACE
1   19 --   {$DECL fUseUnivText}
1   20 --   {$SETC fUseUnivText := TRUE}
1   21 --
1   22 --   USES
1   23 --       {$U libtk/UObject}              UObject,
1   24 --   {$IFC LibraryVersion <= 20}
1   25 --       {$U UFont}                  UFont,
1   26 --   {$ENDC}
1   27 --       {$U QuickDraw}              QuickDraw,
1   28 --       {$U libtk/UDraw}                UDraw,
1   29 --   {$IFC fUseUnivText}
1   30 --       {$U libtk/UUnivText}            UTKUniversalText,
1   31 --   {$ENDC}
1   32 --       {$U UABC}                   UABC;
1   33 --
1   34 --   {$DECL fTextTrace}
1   35 --   {$SETC fTextTrace := fDbgOK}
1   36 --   {$DECL fParaTrace}
1   37 --   {$SETC fParaTrace := fDbgOK}
1   38 --   {$DECL fRngText}
1   39 --   {$SETC fRngText := fDbgOK}
1   40 --
1   41 --   CONST
1   42 --
1   43 --       cVertMargin = 4;
1   44 --       cHorizMargin = 6;
1   45 --
1   46 --       somethingKind = 1;
1   47 --
1   48 --   TYPE
1   49 --
1   50 --       TStyleChange = RECORD
1   51 --           lp:             INTEGER;
1   52 --           newStyle:       TTypeStyle;
1   53 --           END;
1   54 --
1   55 --       TTxtTabDescriptor = RECORD
1   56 --           xCoord:         INTEGER;
1   57 --           quad:           TAlignment;
1   58 --           {MORE LATER}
1   59 --           END;
1   60 --
1   61 --       TDrawAction = (actionDraw, actionInval, actionNone);
1   62 --
1   63 --   { PARAGRAPH SUBCLASSES }
1   64 --
1   65 --       TParaFormat = SUBCLASS OF TObject
1   66 --           dfltTStyle:             TTypeStyle;         {default type style}
1   67 --           wordWrap:               BOOLEAN;
1   68 --           quad:                   TAlignment;
1   69 --           firstIndent:            INTEGER;
1   70 --           leftIndent:             INTEGER;
1   71 --           rightIndent:            INTEGER;
1   72 --           spaceAbovePara:         INTEGER;
1   73 --           spaceBelowPara:         INTEGER;
1   74 --           lineSpacing:            INTEGER;
1   75 --           tabs:                   TArray;
1   76 --
1   77 --           refCount:               INTEGER;        {number of paragraphs referencing this paraFormat}
1   78 --           permanent:              BOOLEAN;        {TRUE -> don't free when refcount goes to zero}
1   79 --           styleSheet:             TStyleSheet;    {NIL if format not in a styleSheet}
1   80 --
1   81 --           FUNCTION TParaFormat.CREATE(object: TObject; heap: THeap; itsStyleSheet: TStyleSheet): TParaFormat;
1   82 --           {$IFC fParaTrace}
1   83 --           PROCEDURE TParaFormat.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1   84 --           {$ENDC}
1   85 --           PROCEDURE TParaFormat.SetTypeStyle(tStyle: TTypeStyle);
1   86 --           PROCEDURE TParaFormat.ChangeRefCountBy(delta: INTEGER);
1   87 --           END;
1   88 --
1   89 --       TParagraph = SUBCLASS OF TString
1   90 --           typeStyles:     TArray; { of TStyleChange }
1   91 --
1   92 --         {Creation/Destruction}
1   93 --           FUNCTION  TParagraph.CREATE(object: TObject; heap: THeap;
1   94 --                                       initialSize: INTEGER; initialTypeStyle: TTypeStyle): TParagraph;
1   95 --           PROCEDURE TParagraph.Free; OVERRIDE;
1   96 --
1   97 --         {Debugging}
1   98 --           {$IFC fParaTrace}
1   99 --           PROCEDURE  TParagraph.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  100 --           {$ENDC}
1  101 --
1  102 --         {Overridden TString methods}
1  103 --           PROCEDURE TParagraph.Draw(i: LONGINT; howMany: INTEGER); OVERRIDE;
1  104 --           FUNCTION  TParagraph.Width(i: LONGINT; howMany: INTEGER): INTEGER; OVERRIDE;
1  105 --
1  106 --         {This method is used by TParagraph.Draw and TParagraph.Width to interpret the typeStyles array}
1  107 --           PROCEDURE TParagraph.DrawLine(startLP, endLP: INTEGER; fDraw: BOOLEAN; fWidth: BOOLEAN;
1  108 --                                         VAR width: INTEGER; VAR styleIndex: INTEGER);
1  109 --
1  110 --         {Type Style Maintainence}
```

```
1  111 --          PROCEDURE TParagraph.ChangeStyle(startLP, endLP: INTEGER; PROCEDURE Change(VAR typeStyle: TTypeStyle);
1  112 --                                                  VAR styleOfStartLP: TTypeStyle);
1  113 --
1  114 --
1  115 --          {These four routines all call ChangeStyle}
1  116 --          PROCEDURE TParagraph.ChgFace(startLP, endLP: INTEGER;
1  117 --                                       newOnFaces: {$IFC LibraryVersion <= 20}TSeteface{$ELSEC}Style{$ENDC};
1  118 --                                                  VAR styleOfStartLP: TTypeStyle);
1  119 --          PROCEDURE TParagraph.ChgFontSize(startLP, endLP: INTEGER; newFontSize: Byte;
1  120 --                                                  VAR styleOfStartLP: TTypeStyle);
1  121 --          PROCEDURE TParagraph.ChgFontFamily(startLP, endLP: INTEGER; newFontFamily: Byte;
1  122 --                                                  VAR styleOfStartLP: TTypeStyle);
1  123 --          PROCEDURE TParagraph.NewStyle(startLP, endLP: INTEGER; newTypeStyle: TTypeStyle);
1  124 --
1  125 --          PROCEDURE TParagraph.CleanRuns;
1  126 --          PROCEDURE TParagraph.UpdateRuns(atLP: INTEGER; replacedChars: INTEGER; insertedChars: INTEGER);
1  127 --
1  128 --        {Character Maintainence}
1  129 --          PROCEDURE TParagraph.ReplPara(fPos, numChars: INTEGER;
1  130 --                                        otherPara: TParagraph; otherFPos, otherNumChars: INTEGER);
1  131 --          PROCEDURE TParagraph.ReplTString(fPos, numChars: INTEGER;
1  132 --                                        otherString: TString; otherFPos, otherNumChars: INTEGER);
1  133 --          PROCEDURE TParagraph.ReplPString(fPos, numChars: INTEGER; pStr: TPString);
1  134 --        {Utilities}
1  135 --        {BuildExtentLRect takes an LPoint that indicates the baseline of the paragraph.  It returns
1  136 --         in extentLRect the bounding rectangle whose height is based on the tallest font in the
1  137 --         paragraph and width is the width of the characters in the paragraph.  Specifically:
1  138 --                        top    := baseLPt.v - tallestFontInfo.ascent;
1  139 --                        bottom := baseLPt.v + tallestFontInfo.descent + tallestFontInfo.leading;
1  140 --                        left   := baseLPt.h;
1  141 --                        right  := baseLpt.h + paragraph.width; }
1  142 --          PROCEDURE TParagraph.BuildExtentLRect(baseLPt: LPoint; VAR extentLRect: LRect);
1  143 --          FUNCTION  TParagraph.FixLP(LP: INTEGER): INTEGER;
1  144 --          PROCEDURE TParagraph.SetTypeStyle(tStyle: TTypeStyle);
1  145 --          PROCEDURE TParagraph.StyleAt(lp: INTEGER; VAR typeStyle: TTypeStyle);
1  146 --
1  147 --        {Word Selection}
1  148 --          PROCEDURE TParagraph.FindWordBounds(orig: INTEGER; VAR first, last: INTEGER);
1  149 --          FUNCTION  TParagraph.Qualifies(pos: INTEGER): BOOLEAN;
1  150 --
1  151 --          END;
1  152 --
1  153 --
1  154 --      {Editable Paragraph}
1  155 --       TEditPara = SUBCLASS OF TParagraph
1  156 --         { character stuff }
1  157 --          bsCount:          INTEGER;
1  158 --         { formatting stuff }
1  159 --          nestLevel:        INTEGER;
1  160 --          format:           TParaFormat;
1  161 --
1  162 --         { paraImage stuff }
1  163 --          beingFiltered:    BOOLEAN;     { TRUE when a type style command has just been
1  164 --                                           performed on this paragraph}
1  165 --
1  166 --          (*
1  167 --          maxImage:         INTEGER;
1  168 --          numImages:        INTEGER;
1  169 --          images:           ARRAY [1..1] OF TParaImage;  {THIS MUST BE LAST FIELD !}
1  170 --          *)
1  171 --          images:           TList;       { Users may subclass TEditPara }
1  172 --
1  173 --         {Creation/Destruction}
1  174 --          FUNCTION  TEditPara.CREATE(object: TObject; heap: THeap; initialSize: INTEGER;
1  175 --                                     itsFormat: TParaFormat): TEditPara;
1  176 --          PROCEDURE TEditPara.Free; OVERRIDE;
1  177 --
1  178 --         {Debugging}
1  179 --          {$IFC fParaTrace}
1  180 --          PROCEDURE TEditPara.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  181 --          {$ENDC}
1  182 --
1  183 --         {Special Editing}
1  184 --          PROCEDURE TEditPara.BeginInsertion(atLP: INTEGER; size: INTEGER);
1  185 --          PROCEDURE TEditPara.EndInsertion;
1  186 --          FUNCTION  TEditPara.GrowSize: INTEGER;
1  187 --          PROCEDURE TEditPara.InsertOneChar(ch: CHAR; atLP: INTEGER);
1  188 --
1  189 --         {Utility}
1  190 --          PROCEDURE TEditPara.SetTypeStyle(tStyle: TTypeStyle); OVERRIDE;
1  191 --
1  192 --         {ParaImage Maintenance}
1  193 --          PROCEDURE TEditPara.EachImage(PROCEDURE ImageProc(paraImage: TParaImage));
1  194 --          PROCEDURE TEditPara.DelImage(delImage: TParaImage; fFree: BOOLEAN);
1  195 --          PROCEDURE TEditPara.InsImage(paraImage: TParaImage);
1  196 --          PROCEDURE TEditPara.DelImgIF(FUNCTION ShouldDelete(paraImage: TParaImage): BOOLEAN);
1  197 --          END;
1  198 --
1  199 --
1  200 --       TLineInfo = SUBCLASS OF TObject
1  201 --           valid:            BOOLEAN;
1  202 --           startLP:          INTEGER;
1  203 --           lastDrawnLP:      INTEGER;  {last character in line to draw: may omit trailing spaces}
1  204 --           endLP:            INTEGER;  {last character in line: equals next lineInfo.startLP - 1}
1  205 --           lineLRect:        LRect;
1  206 --           lineAscent:       INTEGER;
1  207 --
1  208 --           FUNCTION  TLineInfo.CREATE(object: TObject; heap: THeap): TLineInfo;
1  209 --           {$IFC fParaTrace}
1  210 --           PROCEDURE TLineInfo.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  211 --           {$ENDC}
1  212 --
1  213 --         {Used by subclassers who don't like the way the hilite/update
1  214 --          rectangle is chosen so they can overrride it}
1  215 --           FUNCTION TLineInfo.LeftCoord(proposedLeftPixel: LONGINT): LONGINT;
1  216 --           FUNCTION TLineInfo.RightCoord(proposedRightPixel: LONGINT): LONGINT;
1  217 --           END;
1  218 --
1  219 --
1  220 --       TParaImage = SUBCLASS OF TImage
```

```
1  221 --          paragraph:      TEditPara;
1  222 --          height:         INTEGER;    { pixel height of the paragraph}
1  223 --
1  224 --          lineList:       TList;      { of TLineInfo}
1  225 --          changed:        BOOLEAN;
1  226 --          tickCount:      INTEGER;    { incremented (mod MAXINT) every time image is drawn }
1  227 --          startLP:        INTEGER;
1  228 --          endLP:          INTEGER;    { while drawing, this is the LP of the beginning of the next line
1  229 --                                        which, when drawing is finished, may be in another image if the
1  230 --                                        paragraph is split }
1  231 --          textImage:      TTextImage; { the textImage that this image belongs to }
1  232 --          wasOffset:      BOOLEAN;    { used by Building block to determine when to invalidate}
1  233 --
1  234 --        {Creation}
1  235 --          FUNCTION  TParaImage.CREATE(object: TObject; heap: THeap; itsView: TView;
1  236 --                                      itsParagraph: TEditPara; itsLRect: LRect;
1  237 --                                      lineTop: LONGINT; lineLeft: LONGINT): TParaImage;
1  238 --          PROCEDURE TParaImage.Free; OVERRIDE;
1  239 --
1  240 --        {Debugging}
1  241 --          {$IFC fParaTrace}
1  242 --          PROCEDURE TParaImage.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  243 --          {$ENDC}
1  244 --
1  245 --        {Routines}
1  246 --          PROCEDURE TParaImage.ComputeLineInfo(curLine: TLineInfo; maxLineLen: INTEGER;
1  247 --                                               VAR nextLP: INTEGER; VAR lRectNeeded: LRect);
1  248 --          FUNCTION  TParaImage.DfltLineInfo(lineTop: LONGINT; lineLeft: LONGINT): TLineInfo;
1  249 --          PROCEDURE TParaImage.DrawLine(startLP: INTEGER; fDraw: BOOLEAN;
1  250 --                                        stopWidth, wrapWidth: INTEGER;
1  251 --                                        VAR lineWidth, lastToDraw, endLP: INTEGER);
1  252 --          PROCEDURE TParaImage.DrawParaImage(limitLRect: LRect; startLP: INTEGER; drawAction: TDrawAction;
1  253 --                                        invalBits: BOOLEAN; VAR drawnLRect: LRect);
1  254 --          PROCEDURE TParaImage.Draw; OVERRIDE;
1  255 --          PROCEDURE TParaImage.FastDrawLine(startLP, endLP: INTEGER; fDraw: BOOLEAN; fWidth: BOOLEAN;
1  256 --                                        VAR width: INTEGER; VAR styleIndex: INTEGER);
1  257 --          FUNCTION  TParaImage.GetFormat: TParaFormat;
1  258 --          PROCEDURE TParaImage.LineWithLPt(pt: LPoint; VAR lineIndex: INTEGER; VAR lineInfo: TLineInfo);
1  259 --          PROCEDURE TParaImage.LocateLP(LP: INTEGER; VAR lineIndex: INTEGER; VAR pixel: LONGINT);
1  260 --          FUNCTION  TParaImage.LpWithLPt(pt: LPoint): INTEGER;
1  261 --          PROCEDURE TParaImage.OffSetBy(deltaLPt: LPoint); OVERRIDE;
1  262 --          FUNCTION  TParaImage.ParaTextWidth(startLP, endLP: INTEGER): INTEGER;
1  263 --          PROCEDURE TParaImage.RedrawLines(startLine: INTEGER; endLine: INTEGER);
1  264 --          FUNCTION  TParaImage.SeesSameAs(image: TImage): BOOLEAN; OVERRIDE;
1  265 --
1  266 --         {validation/invalidation procs}
1  267 --          PROCEDURE TParaImage.InvalLinesWith(startLP, endLP: INTEGER);
1  268 --          PROCEDURE TParaImage.AdjustLineLPs(atLP, deltaLP: INTEGER);
1  269 --          END;
1  270 --
1  271 --
1  272 --
1  273 --    { MULTI-PARAGRAPH SUBCLASSES }
1  274 --
1  275 --      TStyleSheet = SUBCLASS OF TObject
1  276 --          formats:    TList; {of TParaFormat}
1  277 --
1  278 --        {Creation}
1  279 --          FUNCTION  TStyleSheet.CREATE(object: TObject; heap: THeap): TStyleSheet;
1  280 --          PROCEDURE TStyleSheet.Free; OVERRIDE;
1  281 --
1  282 --        {Installs Default paraFormat into formats list}
1  283 --          PROCEDURE TStyleSheet.InitDefault;
1  284 --        {Debugging}
1  285 --          {$IFC fParaTrace}
1  286 --          PROCEDURE TStyleSheet.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  287 --          {$ENDC}
1  288 --          END;
1  289 --
1  290 --      TTextRange = SUBCLASS OF TObject
1  291 --          firstPara:  TEditPara;
1  292 --          firstIndex: LONGINT;
1  293 --          firstLP:    INTEGER;
1  294 --          lastPara:   TEditPara;
1  295 --          lastIndex:  LONGINT;
1  296 --          lastLP:     INTEGER;
1  297 --
1  298 --        {Creation}
1  299 --          FUNCTION  TTextRange.CREATE(object: TObject; heap: THeap;
1  300 --                                      beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
1  301 --                                      endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER): TTextRange;
1  302 --
1  303 --        {Debugging}
1  304 --          {$IFC fParaTrace}
1  305 --          PROCEDURE TTextRange.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  306 --          {$ENDC}
1  307 --
1  308 --        {AdjustBy adjust the fields of TTextRange by the value of delta (where delta is in LPs)}
1  309 --          PROCEDURE TTextRange.AdjustBy(delta: INTEGER);
1  310 --          END;
1  311 --
1  312 --      TText = SUBCLASS OF TObject
1  313 --          paragraphs: TList;  {of TEditPara }
1  314 --          styleSheet: TStyleSheet;
1  315 --
1  316 --          txtImgList: TList;  {of TTextImages that point to this text;
1  317 --                               IMPORTANT: If the multiple linked textImage feature is used as described in
1  318 --                                          TTextImage below, the application should only store the
1  319 --                                          head text image in this list.  This list is intended for
1  320 --                                          textImages that are viewing the same text object independently
1  321 --                                          (ie in different panels)}
1  322 --
1  323 --        {Creation/Freeing}
1  324 --          FUNCTION  TText.CREATE(object: TObject; heap: THeap; itsStyleSheet: TStyleSheet): TText;
1  325 --
1  326 --          {DfltTextImage can be called after CREATE to create and return a single textImage.  It also
1  327 --           creates one empty paragraph using the first paraFormat in SELF.styleSheet.  It installs the
1  328 --           textImage in txtImgList and the paragraph in paragraphs.  This routine calls
1  329 --           textImage.RecomputeImages to set up the first paraImage.}
1  330 --          FUNCTION  TText.DfltTextImage(view: TView; imageLRect: LRect; imgIsGrouable: BOOLEAN): TTextImage;
```

```
1  331 --
1  332 --          {TText.Free frees all paragraphs that belong to this text object and all textImages that
1  333 --           reference this text object}
1  334 --           PROCEDURE TText.Free; OVERRIDE;
1  335 --           PROCEDURE TText.FreeSelf(freeParas: BOOLEAN);
1  336 --         {Debugging}
1  337 --           {$IFC fParaTrace}
1  338 --           PROCEDURE  TText.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  339 --           {$ENDC}
1  340 --
1  341 --         {Calls to textImage procs get routed through these}
1  342 --           PROCEDURE TText.ChangeSelInOtherPanels(textSelection: TTextSelection);
1  343 --           PROCEDURE TText.DelPara(delPara: TEditPara;  fFree: BOOLEAN);
1  344 --           PROCEDURE TText.Draw;
1  345 --           PROCEDURE TText.HiliteRange(highTransit: THighTransit; textRange: TTextRange; wholePara: BOOLEAN);
1  346 --           PROCEDURE TText.HiliteParagraphs(highTransit: THighTransit;
1  347 --                                            startIndex: LONGINT; startLP: INTEGER;
1  348 --                                            endIndex: LONGINT; endLP: INTEGER; wholePara: BOOLEAN);
1  349 --           PROCEDURE TText.InsParaAfter(existingPara: TEditPara;  newPara: TEditPara);
1  350 --           PROCEDURE TText.Invalidate;
1  351 --           PROCEDURE TText.MarkChanged(textRange: TTextRange);
1  352 --           PROCEDURE TText.RecomputeImages;
1  353 --           FUNCTION  TText.SelectAll(textImage: TTextImage): TTextSelection;
1  354 --           END;
1  355 --
1  356 --
1  357 --
1  358 --       TTextImage = SUBCLASS OF TImage
1  359 --           text:                TText;       {complete list of paragraphs}
1  360 --           imageList:           TList;       {paraImages for some range of paragraphs in text}
1  361 --           tickCount:           INTEGER;
1  362 --           growsDynamically:    BOOLEAN;     {TRUE -->  extentLRect bottom grows as more text entered;
1  363 --                                              FALSE -->  text is truncated at last line that fits}
1  364 --           minHeight:           INTEGER;     {the minimum height to shrink (if growsDynamically=TRUE);
1  365 --                                              defaults to height of original extentLRect}
1  366 --
1  367 --           formerBottom:        LONGINT;     {Used by Invalidate when the displayed paragraphs get shorter
1  368 --                                              and text at end needs to be erased}
1  369 --           updateLRect:         LRect;       {"                 "                    "}
1  370 --
1  371 --           firstLinePixel:      LONGINT;     {Used by Text BB to limit what gets erased on first update line}
1  372 --           useFirstPixel:       BOOLEAN;
1  373 --
1  374 --
1  375 --           { The following fields support multiple linked text images displaying a single text object,
1  376 --             where the text "flows" from one box to the next.  APPLICATIONS ARE RESPONSIBLE FOR
1  377 --             MAINTAINING THESE FIELDS.  This Building Block uses these fields for drawing, etc.
1  378 --             All text images in a chain should have growsDynamically set to FALSE (except possibly
1  379 --             for the last text image in a chain).
1  380 --             For applications that DO NOT use this feature, the fields will always be as follows:
1  381 --                   startLP = 0;
1  382 --                   endLP =   LP of last character in last paragraph; (if growsDynamically = TRUE)
1  383 --                             LP of last character that fit in extentLRect; (if growsDynamically = FALSE)
1  384 --                   prevTxtImg, nextTxtImg = NIL;
1  385 --                   headTxtImg = SELF;
1  386 --                   tailTxtImg = SELF;
1  387 --           }
1  388 --           firstIndex:      LONGINT;     {index of paragraph at SELF.imageList.First}
1  389 --           startLP:         INTEGER;     {startLP of paragraph at SELF.imageList.First}
1  390 --           endLP:           INTEGER;     {endLP of paragraph at SELF.imageList.Last}
1  391 --
1  392 --           prevTxtImg:      TTextImage;  {  for linking textImages that display different parts of }
1  393 --           nextTxtImg:      TTextImage;  {  the same text object, eg: columns}
1  394 --           headTxtImg:      TTextImage;  {points to first text image in this list}
1  395 --           tailTxtImg:      TTextImage;  {points to last text image in this list}
1  396 --
1  397 --         {Creation}
1  398 --           FUNCTION  TTextImage.CREATE(object: TObject; heap: THeap; itsView: TView;
1  399 --                                       itsLRect: LRect; itsText: TText; isGrowable: BOOLEAN): TTextImage;
1  400 --
1  401 --           {TTextImage.Free frees all text images and their paraImages in the text image chain.
1  402 --            It does NOT free any paragraphs, text objects, or paraFormats.  Call this only once
1  403 --            for each text image chain (NOT for each text image in the chain).  Note that TText.Free
1  404 --            frees its textImages so calling this routine is not necessary in most cases}
1  405 --           PROCEDURE  TTextImage.Free; OVERRIDE;
1  406 --
1  407 --           {TTextImage.FreeOneTextImage frees just one text image from the chain.  It pays no attention
1  408 --            to links or whether this is the head text image.  Maintenance of these fields must be
1  409 --            handled by the caller before calling this routine.  Those who do not use linked text images
1  410 --            should always call TTextImage.Free above, NOT this routine}
1  411 --           PROCEDURE TTextImage.FreeOneTextImage;
1  412 --
1  413 --         {Debugging}
1  414 --           {$IFC fParaTrace}
1  415 --           PROCEDURE TTextImage.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  416 --           {$ENDC}
1  417 --
1  418 --         {Drawing}
1  419 --           PROCEDURE TTextImage.Draw; OVERRIDE;
1  420 --           PROCEDURE TTextImage.DrawImages(fDraw: BOOLEAN);
1  421 --           PROCEDURE TTextImage.DrawOrInval(invalBits: BOOLEAN);
1  422 --           PROCEDURE TTextImage.HiliteText(highTransit: THighTransit;
1  423 --                                           startIndex: LONGINT; startLP: INTEGER;
1  424 --                                           endIndex: LONGINT; endLP: INTEGER; wholePara: BOOLEAN);
1  425 --
1  426 --         {Locating}
1  427 --           PROCEDURE TTextImage.FindParaAndLp(LPt: LPoint; VAR paraImage: TParaImage;
1  428 --                                              VAR paraIndex: LONGINT; VAR aLP: INTEGER);
1  429 --           FUNCTION  TTextImage.FindTextImage(VAR mouseLPt: LPoint; VAR firstTxtImg: TTextImage): TTextImage;
1  430 --           FUNCTION  TTextImage.ImageBottom: LONGINT;
1  431 --           PROCEDURE TTextImage.GetImageRange(firstIndex: LONGINT; VAR firstLP: INTEGER;
1  432 --                                              lastIndex: LONGINT; VAR lastLP: INTEGER;
1  433 --                                              VAR firstImage, lastImage: TParaImage);
1  434 --           FUNCTION  TTextImage.ImageWith(paragraph: TEditPara; lp: INTEGER): TParaImage;
1  435 --           PROCEDURE TTextImage.MousePress(mouseLPt: LPoint); OVERRIDE;
1  436 --           PROCEDURE TTextImage.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  437 --
1  438 --         {Image maintenence}
1  439 --           PROCEDURE TTextImage.AddImage(paraImage: TParaImage);
1  440 --           PROCEDURE TTextImage.DelImagesWith(delPara: TEditPara);
```

```
1  441 --          PROCEDURE TTextImage.InsertNewPara(existingPara, newPara: TEditPara);
1  442 --          PROCEDURE TTextImage.InvalAll;
1  443 --          PROCEDURE TTextImage.Invalidate; OVERRIDE; {Invalidate changed lineLRects in changed paraimages}
1  444 --          PROCEDURE TTextImage.MarkChanged(startIndex: LONGINT; startLP: INTEGER;
1  445 --                                           endIndex: LONGINT; endLP: INTEGER);
1  446 --          FUNCTION  TTextImage.NewTextSelection(firstPara: TEditPara; firstIndex: LONGINT; firstLP: INTEGER;
1  447 --                                                lastPara: TEditPara; lastIndex: LONGINT; lastLP: INTEGER
1  448 --                                                ): TTextSelection;
1  449 --          PROCEDURE TTextImage.RecomputeImages(drawAction: TDrawAction; invalBits: BOOLEAN);
1  450 --          PROCEDURE TTextImage.Resize(newExtent: LRect); OVERRIDE;
1  451 --          FUNCTION  TTextImage.SeesSameAs(image: TImage): BOOLEAN; OVERRIDE;
1  452 --
1  453 --        {By default SetFirstIndex just sets firstIndex to 0, but subclassers may override this
1  454 --         if they want the display to start from other than the first paragraph}
1  455 --          PROCEDURE TTextImage.SetFirstIndex;
1  456 --
1  457 --          {These routines are provided so that users can subclass the appropriate class and
1  458 --           then override these methods so that the building block will create the user's subclass
1  459 --           when generating new instances of that class. }
1  460 --          FUNCTION  TTextImage.NewEditPara(initialSize: INTEGER; itsFormat: TParaFormat): TEditPara;
1  461 --          FUNCTION  TTextImage.NewParaImage(itsParagraph: TEditPara; itsLRect: LRect;
1  462 --                                            lineTop: LONGINT; lineLeft: LONGINT): TParaImage;
1  463 --          FUNCTION  TTextImage.NewTextImage(heap: THeap; itsView: TView; itsLRect: LRect;
1  464 --                                            itsText: TText; isGrowable: BOOLEAN): TTextImage;
1  465 --          FUNCTION  TTextImage.TxtImgForClipBoard(heap: THeap; itsView: TView; itsLRect: LRect;
1  466 --                                            itsText: TText; isGrowable: BOOLEAN): TTextImage;
1  467 --          END;
1  468 --
1  469 --
1  470 --        {Clipboard Text View}
1  471 --        TTextView = SUBCLASS OF TView
1  472 --            textImage:   TTextImage;
1  473 --            valid:       BOOLEAN;        {If FALSE, calls Recompute before Drawing}
1  474 --
1  475 --          {Creation}
1  476 --            FUNCTION  TTextView.CREATE(object: TObject; heap: THeap; itsPanel: TPanel; itsExtent: LRect)
1  477 --                                       : TTextView;
1  478 --
1  479 --          {Debugging}
1  480 --            {$IFC fParaTrace}
1  481 --            PROCEDURE TTextView.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  482 --            {$ENDC}
1  483 --
1  484 --            {$IFC fUseUnivText}
1  485 --            PROCEDURE TTextView.CreateUniversalText; OVERRIDE;
1  486 --            {$ENDC}
1  487 --            PROCEDURE TTextView.Draw; OVERRIDE;
1  488 --            PROCEDURE TTextView.MousePress(mouseLPt: LPoint); OVERRIDE;
1  489 --            END;
1  490 --
1  491 --
1  492 --        {$IFC fUseUnivText}
1  493 --        TTextWriteUnivText = SUBCLASS OF TTKWriteUnivText
1  494 --            textSelection:   TTextSelection;
1  495 --            currIndex:       LONGINT;
1  496 --            currPara:        TEditPara;
1  497 --            currLP:          INTEGER;
1  498 --            currStyleIndex:  INTEGER;
1  499 --            currTStyles:      TArray;
1  500 --          {Creation}
1  501 --            FUNCTION  TTextWriteUnivText.CREATE(object: TObject; heap: THeap;
1  502 --                                                itsString: TString; itsDataSize: INTEGER;
1  503 --                                                itsTextSel: TTextSelection): TTextWriteUnivText;
1  504 --
1  505 --          {Debugging}
1  506 --            {$IFC fParaTrace}
1  507 --            PROCEDURE TTextWriteUnivText.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  508 --            {$ENDC}
1  509 --
1  510 --            PROCEDURE TTextWriteUnivText.FillParagraph; OVERRIDE;
1  511 --            END;
1  512 --        {$ENDC}
1  513 --
1  514 --
1  515 --        TTextSelection = SUBCLASS OF TSelection
1  516 --            textImage:          TTextImage;
1  517 --            textRange:          TTextRange;
1  518 --            isWordSelection:    BOOLEAN;
1  519 --            isParaSelection:    BOOLEAN;
1  520 --            viewTick:           INTEGER;
1  521 --            amTyping:           BOOLEAN;
1  522 --            currTypeStyle:      TTypeStyle;
1  523 --
1  524 --            FUNCTION  TTextSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
1  525 --                                            itsTextImage: TTextImage; itsAnchorLPt: LPoint;
1  526 --                                            beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
1  527 --                                            endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER
1  528 --                                            ): TTextSelection;
1  529 --
1  530 --
1  531 --          {Debugging}
1  532 --            {$IFC fParaTrace}
1  533 --            PROCEDURE TTextSelection.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  534 --            {$ENDC}
1  535 --
1  536 --          {Commands}
1  537 --            PROCEDURE TTextSelection.KeyText;
1  538 --            FUNCTION  TTextSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  539 --            FUNCTION  TTextSelection.NewStyleCmd(heap: THeap; cmdNumber: TCmdNumber;
1  540 --                                                 textImage: TTextImage): TCommand;
1  541 --            FUNCTION  TTextSelection.NewCutCopyCmd(heap: THeap; cmdNumber: TCmdNumber;
1  542 --                                                 textImage: TTextImage): TCommand; DEFAULT;
1  543 --            PROCEDURE TTextSelection.StyleFromContext; DEFAULT;
1  544 --            PROCEDURE TTextSelection.DoChangeStyle(cmdNumber: TCmdNumber; paragraph: TParagraph;
1  545 --                                                   firstLP: INTEGER; lastLP: INTEGER; VAR newStyle: TTypeStyle);
1  546 --            PROCEDURE TTextSelection.ChangeStyle(cmdNumber: TCmdNumber); DEFAULT;
1  547 --          {Editing}
1  548 --            PROCEDURE TTextSelection.ChangeText(PROCEDURE TextEdit; PROCEDURE Adjust); DEFAULT;
1  549 --            FUNCTION  TTextSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection; DEFAULT;
1  550 --            PROCEDURE TTextSelection.CutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN); DEFAULT;
```

```
1  551 --          PROCEDURE TTextSelection.DeleteAndFree; DEFAULT;
1  552 --          FUNCTION  TTextSelection.DeleteButSave: TText; DEFAULT;
1  553 --
1  554 --        {Highlighting}
1  555 --          PROCEDURE TTextSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1  556 --
1  557 --        {Selecting}
1  558 --          FUNCTION  TTextSelection.BecomeInsertionPoint: TInsertionPoint;
1  559 --          PROCEDURE TTextSelection.GetHysteresis(VAR hysterPt: Point); OVERRIDE;
1  560 --          PROCEDURE TTextSelection.MousePress(mouseLPt: LPoint); OVERRIDE;
1  561 --          FUNCTION  TTextSelection.SelSize: INTEGER; ABSTRACT;
1  562 --
1  563 --        {Invalidation}
1  564 --          PROCEDURE TTextSelection.Invalidate; DEFAULT;
1  565 --
1  566 --        {Generate Text Selection in another panel (ie. another Text Image)}
1  567 --          FUNCTION  TTextSelection.ReplicateForOtherPanel(itsTextImage: TTextImage): TTextSelection;
1  568 --          END;
1  569 --
1  570 --
1  571 --      TInsertionPoint = SUBCLASS OF TTextSelection
1  572 --          typingCmd:          TTypingCmd;     {the current typing command (if user is typing)}
1  573 --          styleCmdNumber:     INTEGER;        {Set to cmdNumber when a type style item is chosen,
1  574 --                                                  set to zero otherwise}
1  575 --          newestLP:           INTEGER;        {the lp position as updated between KeyPause's}
1  576 --          justReturned:       BOOLEAN;        {flag that prevents redundant update in KeyPause}
1  577 --
1  578 --          nextHighTransit:    THighTransit;
1  579 --          nextTransitTime:    LONGINT;
1  580 --
1  581 --        {Creation/Freeing}
1  582 --          FUNCTION  TInsertionPoint.CREATE(object: TObject; heap: THeap; itsView: TView;
1  583 --                                           itsTextImage: TTextImage; itsAnchorLPt: LPoint; itsParagraph: TEditPara;
1  584 --                                           itsIndex: LONGINT; itsLP: INTEGER): TInsertionPoint;
1  585 --
1  586 --        {Debugging}
1  587 --          {$IFC fParaTrace}
1  588 --          PROCEDURE TInsertionPoint.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  589 --          {$ENDC}
1  590 --
1  591 --        {Commands}
1  592 --          PROCEDURE TInsertionPoint.IdleBegin(centiSeconds: LONGINT); OVERRIDE;
1  593 --          PROCEDURE TInsertionPoint.IdleContinue(centiSeconds: LONGINT); OVERRIDE;
1  594 --          PROCEDURE TInsertionPoint.IdleEnd(centiSeconds: LONGINT); OVERRIDE;
1  595 --          FUNCTION  TInsertionPoint.NewCutCopyCmd(heap: THeap; cmdNumber: TCmdNumber;
1  596 --                                                  textImage: TTextImage): TCommand; OVERRIDE;
1  597 --          PROCEDURE TInsertionPoint.StyleFromContext; OVERRIDE;
1  598 --
1  599 --        {Editing}
1  600 --          PROCEDURE TInsertionPoint.CutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN); OVERRIDE;
1  601 --          PROCEDURE TInsertionPoint.FinishPaste(clipSelection: TSelection; pic: PicHandle);
1  602 --          PROCEDURE TInsertionPoint.InsertText(text: TText; isParaSelection: BOOLEAN; isWordSelection: BOOLEAN;
1  603 --                                                                                      universalText: BOOLEAN);
1  604 --          PROCEDURE TInsertionPoint.KeyBack(fWord: BOOLEAN); OVERRIDE;
1  605 --          PROCEDURE TInsertionPoint.KeyChar(ch: CHAR); OVERRIDE;
1  606 --          PROCEDURE TInsertionPoint.KeyClear; OVERRIDE;
1  607 --          PROCEDURE TInsertionPoint.KeyForward(fWord: BOOLEAN); OVERRIDE;
1  608 --
1  609 --        {Selecting}
1  610 --          PROCEDURE TInsertionPoint.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  611 --          PROCEDURE TInsertionPoint.MousePress(mouseLPt: LPoint); OVERRIDE;
1  612 --          PROCEDURE TInsertionPoint.MouseRelease; OVERRIDE;
1  613 --
1  614 --          END;
1  615 --
1  616 --
1  617 --      TOneParaSelection = SUBCLASS OF TTextSelection
1  618 --          anchorBegin:    INTEGER;
1  619 --          anchorEnd:      INTEGER;    {anchorBegin <> anchorEnd iff double or triple click}
1  620 --
1  621 --        {Creation/Freeing}
1  622 --          FUNCTION  TOneParaSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
1  623 --                                             itsTextImage: TTextImage; itsAnchorLPt: LPoint; itsParagraph: TEditPara;
1  624 --                                             itsIndex: LONGINT; oldLP: INTEGER; currLP: INTEGER): TOneParaSelection;
1  625 --
1  626 --        {Debugging}
1  627 --          {$IFC fParaTrace}
1  628 --          PROCEDURE TOneParaSelection.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  629 --          {$ENDC}
1  630 --
1  631 --        {Commands}
1  632 --          PROCEDURE TOneParaSelection.StyleFromContext; OVERRIDE;
1  633 --
1  634 --        {Editing}
1  635 --          FUNCTION  TOneParaSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection; OVERRIDE;
1  636 --          PROCEDURE TOneParaSelection.DeleteAndFree; OVERRIDE;
1  637 --          FUNCTION  TOneParaSelection.DeleteButSave: TText; OVERRIDE;
1  638 --
1  639 --        {Selecting}
1  640 --          PROCEDURE TOneParaSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  641 --          PROCEDURE TOneParaSelection.MouseRelease; OVERRIDE;
1  642 --
1  643 --          END;
1  644 --
1  645 --
1  646 --      TMultiParaSelection = SUBCLASS OF TTextSelection
1  647 --          anchorPara:     TEditPara;
1  648 --          anchorIndex:    LONGINT;
1  649 --          anchorBegin:    INTEGER;
1  650 --          anchorEnd:      INTEGER;    {anchorBegin <> anchorEnd iff double or triple click}
1  651 --
1  652 --        {Creation/Freeing}
1  653 --          FUNCTION  TMultiParaSelection.CREATE(object: TObject; heap: THeap; itsView: TView;
1  654 --                                               itsTextImage: TTextImage; itsAnchorLPt: LPoint;
1  655 --                                               beginPara: TEditPara; beginIndex: LONGINT; beginLP: INTEGER;
1  656 --                                               endPara: TEditPara; endIndex: LONGINT; endLP: INTEGER;
1  657 --                                               beginIsAnchor: BOOLEAN): TMultiParaSelection;
1  658 --
1  659 --        {Debugging}
1  660 --          {$IFC fParaTrace}
```

```
1  661 --            PROCEDURE TMultiParaSelection.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  662 --            {$ENDC}
1  663 --
1  664 --          {Commands}
1  665 --            PROCEDURE TMultiParaSelection.StyleFromContext; OVERRIDE;
1  666 --
1  667 --          {Editing}
1  668 --            FUNCTION  TMultiParaSelection.CopySelf(heap: THeap; view: TView): TMultiParaSelection; OVERRIDE;
1  669 --            FUNCTION  TMultiParaSelection.Delete(saveIt: BOOLEAN): TText;
1  670 --            PROCEDURE TMultiParaSelection.DeleteAndFree; OVERRIDE;
1  671 --            FUNCTION  TMultiParaSelection.DeleteButSave: TText; OVERRIDE;
1  672 --
1  673 --          {Selecting}
1  674 --            PROCEDURE TMultiParaSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  675 --            PROCEDURE TMultiParaSelection.MouseRelease; OVERRIDE;
1  676 --
1  677 --            END;
1  678 --
1  679 --
1  680 --                          {-------------  COMMANDS  -----------------}
1  681 --
1  682 --        TClearTextCmd = SUBCLASS OF TCommand
1  683 --
1  684 --          {Variables}
1  685 --            savedText:        TText;       {save the cleared text for undo}
1  686 --            text:             TText;       {the text object we are clearing}
1  687 --
1  688 --          {Creation}
1  689 --            FUNCTION  {TClearTextCmd.}CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
1  690 --                                             itsImage: TImage; itsText: TText): TClearTextCmd;
1  691 --
1  692 --            PROCEDURE TClearTextCmd.Free; OVERRIDE;
1  693 --            {$IFC fParaTrace}
1  694 --            PROCEDURE TClearTextCmd.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  695 --            {$ENDC}
1  696 --          {Command Execution}
1  697 --            PROCEDURE TClearTextCmd.Commit; OVERRIDE;
1  698 --            PROCEDURE TClearTextCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  699 --            END;
1  700 --
1  701 --        TStyleCmd = SUBCLASS OF TCommand
1  702 --
1  703 --          {Variables}
1  704 --            text:                 TText;
1  705 --            textSelection:        TTextSelection;
1  706 --            firstFiltParaIndex:   LONGINT;
1  707 --            lastFiltParaIndex:    LONGINT;
1  708 --            filtFirstLP:          INTEGER;
1  709 --            filtLastLP:           INTEGER;
1  710 --            currFilteredPara:     TEditPara;       {handle to most recently filtered paragraph}
1  711 --            filteredStyles:       TArray;          {changed type styles of most recently filtered paragraph}
1  712 --
1  713 --          {Creation}
1  714 --            FUNCTION  TStyleCmd.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
1  715 --                                       itsImage: TImage;
1  716 --                                       itsFirstIndex: LONGINT; itsLastIndex:  LONGINT;
1  717 --                                       itsLPFirst: INTEGER; itsLPLast: INTEGER;
1  718 --                                       itsSelection: TTextSelection): TStyleCmd;
1  719 --
1  720 --            PROCEDURE TStyleCmd.Free; OVERRIDE;
1  721 --            {$IFC fParaTrace}
1  722 --            PROCEDURE TStyleCmd.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  723 --            {$ENDC}
1  724 --          {Command Execution}
1  725 --            PROCEDURE TStyleCmd.Commit; OVERRIDE;
1  726 --            PROCEDURE TStyleCmd.FilterAndDo(actualObject: TObject;
1  727 --                                            PROCEDURE DoToObject(filteredObject: TObject)); OVERRIDE;
1  728 --            PROCEDURE TStyleCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  729 --            END;
1  730 --
1  731 --        TTextCutCopy = SUBCLASS OF TCutCopyCommand
1  732 --
1  733 --          {Variables}
1  734 --            text:    TText;
1  735 --
1  736 --          {Creation}
1  737 --            FUNCTION  TTextCutCopy.CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
1  738 --                                          itsImage: TImage;
1  739 --                                          isCutCmd: BOOLEAN; itsText: TText): TTextCutCopy;
1  740 --
1  741 --            PROCEDURE TTextCutCopy.Free; OVERRIDE;
1  742 --            {$IFC fParaTrace}
1  743 --            PROCEDURE TTextCutCopy.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  744 --            {$ENDC}
1  745 --          {Command Execution}
1  746 --            PROCEDURE TTextCutCopy.DoCutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN;
1  747 --                                             cmdPhase: TCmdPhase); OVERRIDE;
1  748 --            END;
1  749 --
1  750 --        TTextPaste = SUBCLASS OF TPasteCommand
1  751 --
1  752 --          {Variables}
1  753 --            savedText:   TText;
1  754 --            pasteRange:  TTextRange;      {The text range spanned by the pasted text}
1  755 --            text:        TText;
1  756 --            origIsPara:  BOOLEAN;
1  757 --            origIsWord:  BOOLEAN;
1  758 --            clipIsPara:  BOOLEAN;
1  759 --
1  760 --          {Creation}
1  761 --            FUNCTION  TTextPaste.CREATE(object: TObject; heap: THeap;  itsImage: TImage;
1  762 --                                        itsText: TText): TTextPaste;
1  763 --
1  764 --            PROCEDURE TTextPaste.Free; OVERRIDE;
1  765 --            {$IFC fParaTrace}
1  766 --            PROCEDURE TTextPaste.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  767 --            {$ENDC}
1  768 --          {Command Execution}
1  769 --            PROCEDURE TTextPaste.Commit; OVERRIDE;
1  770 --            PROCEDURE TTextPaste.DoPaste(clipSelection: TSelection; pic: PicHandle; cmdPhase: TCmdPhase);
```

```
1  771 --                                                                                                  OVERRIDE;
1  772 --          END;
1  773 --
1  774 --
1  775 --      TTypingCmd = SUBCLASS OF TCommand
1  776 --
1  777 --        {Variables}
1  778 --          savedText:              TText;
1  779 --          text:                   TText;
1  780 --          newCharCount:           INTEGER;
1  781 --          newParaCount:           INTEGER;
1  782 --          typingRange:            TTextRange;     {The text range spanned by the typed characters}
1  783 --          otherInsPts:            TList;
1  784 --
1  785 --        {Creation}
1  786 --          FUNCTION  TTypingCmd.CREATE(object: TObject; heap: THeap; itsImage: TImage;
1  787 --                                      itsText: TText): TTypingCmd;
1  788 --
1  789 --          PROCEDURE TTypingCmd.Free; OVERRIDE;
1  790 --          {$IFC fParaTrace}
1  791 --          PROCEDURE TTypingCmd.Fields(PROCEDURE Field(nameAndType: S255)); OVERRIDE;
1  792 --          {$ENDC}
1  793 --        {Command Execution}
1  794 --          PROCEDURE TTypingCmd.Commit; OVERRIDE;
1  795 --          PROCEDURE TTypingCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  796 --          END;
1  797 --
1  798 --
1  799 --    VAR fParaTrace:               BOOLEAN;
1  800 --        fTextTrace:               BOOLEAN;
1  801 --
1  802 --
1  803 --    IMPLEMENTATION
1  804 --
1  805 --    (*
1  806 --    {$I UTEXT2.text}  {Paragraph classes}
1  807 --    {$I UTEXT3.text}  {TStyleSheet, TText, TTextImage, TTextView}
1  808 --    {$I UTEXT4.text}  {Text Selections and Commands}
1  809 --    *)
1  810 --
2    1 --
2    2 --
1  811 --    {$I LibTK/UTEXT2.text}  {Paragraph classes}
3    1 --
3    2 --
1  812 --    {$I LibTK/UTEXT3.text}  {TStyleSheet, TText, TTextImage, TTextView}
4    1 --
4    2 --
1  813 --    {$I LibTK/UTEXT4.text}  {Text Selections and Commands}
1  814 --    END.
1  815 --
```

1. libtk/utext.TEXT
2. LibTK/UTEXT2.text
3. LibTK/UTEXT3.text
4. LibTK/UTEXT4.text

```
  RedrawLines          263•( 1)
  refCount              77•( 1)
  ReplicateForOthe     567•( 1)
  ReplPara             129•( 1)
  ReplPString          133•( 1)
  ReplTString          131•( 1)
  Resize               450•( 1)
  RightCoord           216•( 1)
  rightIndent           71•( 1)

-S-
  savedText            685 ( 1)  753 ( 1)  778 ( 1)
  SeesSameAs           264•( 1)  451•( 1)
  SelectAll            353•( 1)
  SelSize              561•( 1)
  SetFirstIndex        455•( 1)
  SetTypeStyle          85•( 1)  144•( 1)  190•( 1)
  somethingKind         46•( 1)
  spaceAboveePara       72•( 1)
  spaceBelowPara        73•( 1)
  startLP              202•( 1)  227•( 1)  389•( 1)
  StyleAt              145•( 1)
  styleCmdNumber       573•( 1)
  StyleFromContext     543•( 1)  597•( 1)  632•( 1)  665•( 1)
  styleSheet            79•( 1)  314•( 1)

-T-
  tabs                  75•( 1)
  tailTxtImg           395•( 1)
  TAlignment            57 ( 1)   68 ( 1)
  TArray                75 ( 1)   90 ( 1)  499 ( 1)  711 ( 1)
  TClearTextCmd        682•( 1)  690 ( 1)
  TCommand             538 ( 1)  540 ( 1)  542 ( 1)  596 ( 1)  682 ( 1)  701 ( 1)  775 ( 1)
  TCutCopyCommand      731 ( 1)
  TDrawAction           61•( 1)
  TEditPara            155•( 1)  175 ( 1)  221 ( 1)  291 ( 1)  294 ( 1)  460 ( 1)  496 ( 1)  647 ( 1)  710 ( 1)
  text                 359 ( 1)  686•( 1)  704 ( 1)  734 ( 1)  755•( 1)  779•( 1)
  textImage            231•( 1)  472 ( 1)  516 ( 1)
  textRange            517•( 1)
  textSelection        494 ( 1)  705•( 1)
  THighTransit         578 ( 1)
  tickCount            226•( 1)  361•( 1)
  TImage               220 ( 1)  358 ( 1)
  TInsertionPoint      558 ( 1)  571•( 1)  584 ( 1)
  TLineInfo            200•( 1)  208 ( 1)  248 ( 1)
  TList                171 ( 1)  224 ( 1)  276 ( 1)  313 ( 1)  316 ( 1)  360 ( 1)  783 ( 1)
  TMultiParaSelect     549 ( 1)  635 ( 1)  646•( 1)  657 ( 1)  668 ( 1)
  TObject               65 ( 1)  200 ( 1)  275 ( 1)  290 ( 1)  312 ( 1)
  TOneParaSelectio     617•( 1)  624 ( 1)
  TParaFormat           65•( 1)   81 ( 1)  160 ( 1)  257 ( 1)
  TParagraph            89•( 1)   94 ( 1)  155 ( 1)
  TParaImage           220•( 1)  237 ( 1)  434 ( 1)  462 ( 1)
  TPasteCommand        750 ( 1)
  TSelection           515 ( 1)
  TString               89 ( 1)
  TStyleChange          50•( 1)
  TStyleCmd            701•( 1)  718 ( 1)
  TStyleSheet           79 ( 1)  275•( 1)  279 ( 1)  314 ( 1)
  TText                312•( 1)  324 ( 1)  359 ( 1)  552 ( 1)  637 ( 1)  669 ( 1)  671 ( 1)  685 ( 1)  686 ( 1)  704 ( 1)
                       734 ( 1)  753 ( 1)  755 ( 1)  778 ( 1)  779 ( 1)
  TTextCutCopy         731•( 1)  739 ( 1)
  TTextImage           231 ( 1)  330 ( 1)  358•( 1)  392 ( 1)  393 ( 1)  394 ( 1)  395 ( 1)  399 ( 1)  429 ( 1)  464 ( 1)
                       466 ( 1)  472 ( 1)  516 ( 1)
  TTextPaste           750•( 1)  762 ( 1)
  TTextRange           290•( 1)  301 ( 1)  517 ( 1)  754 ( 1)  782 ( 1)
  TTextSelection       353 ( 1)  448 ( 1)  494 ( 1)  515•( 1)  528 ( 1)  567 ( 1)  571 ( 1)  617 ( 1)  646 ( 1)  705 ( 1)
  TTextView            471•( 1)  477 ( 1)
  TTextWriteUnivTe     493•( 1)  503 ( 1)
  TTKWriteUnivText     493 ( 1)
  TTxtTabDescripto      55•( 1)
  TTypeStyle            52 ( 1)   66 ( 1)  522 ( 1)
  TTypingCmd           572 ( 1)  775•( 1)  787 ( 1)
  TView                471 ( 1)
  TxtImgForClipBoa     465•( 1)
  txtImgList           316•( 1)
  typeStyles            90 ( 1)
  typingCmd            572 ( 1)
  typingRange          782•( 1)

-U-
  UABC                  32•( 1)
  UDraw                 28•( 1)
  UFont                 25•( 1)
  UObject               23•( 1)
  updateLRect          369•( 1)
  UpdateRuns           126•( 1)
  useFirstPixel        372•( 1)
  UText                  1•( 1)
  UTKUniversalText      30•( 1)

-V-
  valid                201 ( 1)  473•( 1)
  viewTick             520•( 1)

-W-
  wasOffset            232•( 1)
  Width                104•( 1)
  wordWrap              67•( 1)

-X-
  xCoord                56•( 1)

••• End Xref: 264 id's   468 references      [421880 bytes/4735 id's/43265 refs]
```

```
1    1 --  {$E+}
1    2 --  {$E ERR1.TEXT}
1    3 --  {-------------------------------------------------------------------------------}
1    4 --  {                                 UUniversalText V0.3                           }
1    5 --  {-------------------------------------------------------------------------------}
1    6 --  {$SETC ForOS := TRUE }
1    7 --  {$DECL WithUObject}
1    8 --  {$SETC WithUObject := TRUE}                {Note: TRUE/FALSE status MUST agree with below}
1    9 --
1   10 --
1   11 --
1   12 --  UNIT {$IFC WithUObject}
1   13 --    UTKUniversalText
1   14 --       {$ELSEC}
1   15 --       UUniversalText
1   16 --       {$ENDC};
1   17 --
1   18 --  {$DECL IsIntrinsic}
1   19 --  {$SETC IsIntrinsic := TRUE}
1   20 --
1   21 --  {$SETC WithUObject := TRUE}                {Note: TRUE/FALSE status MUST agree with above}
1   22 --
1   23 --  {$IFC IsIntrinsic}
1   24 --  INTRINSIC;
1   25 --  {$ENDC}
1   26 --
1   27 --  {$IFC NOT WithUObject}
1   28 --      {$SETC LibraryVersion := 30 } { 10 = 1.0 libraries; 13 = 1.3 libraries; 20 = Pepsi, 30 = Spring, etc. }
1   29 --  {$ENDC}
1   30 --
1   31 --  INTERFACE
1   32 --
1   33 --  USES
1   34 --  {$IFC WithUObject}
1   35 --      {$U libtk/UObject}          UObject,
1   36 --      {$SETC fTrce := fTrace}
1   37 --  {$ENDC}
1   38 --      {$U libsm/UnitStd.obj }     UnitStd,
1   39 --      {$U libsm/UnitHz.obj }      UnitHz,
1   40 --  {$IFC NOT WithUObject}
1   41 --      {$U libpl/UClascal}               UClascal,          {Will be in PASLIB in Spring}
1   42 --      {$U libqd/Storage.obj }     Storage,
1   43 --  {$ENDC}
1   44 --  {$IFC LibraryVersion <= 20}
1   45 --      {$U libfm/FontMgr.obj }     FontMgr,
1   46 --      {$U libqd/QuickDraw.obj }   QuickDraw,
1   47 --  {$ELSEC}
1   48 --      {$U libqd/QuickDraw.obj }   QuickDraw,
1   49 --      {$U libfm/FontMgr.obj }     FontMgr,
1   50 --  {$ENDC}
1   51 --      {$U libos/SysCall.obj }     Syscall,
1   52 --      {$U libpm/PMDecl.obj }      PMDecl,
1   53 --      {$U libpr/PrStdInfo.obj }   PrStdInfo,
1   54 --      {$U libsu/UnitFmt.obj }     UnitFmt,
1   55 --      {$U libsu/UnitCS.obj }      UnitCS,
1   56 --      {$U libwm/Events.obj }      Events,
1   57 --      {$U libsu/Scrap.obj }       Scrap;
1   58 --
1   59 --  {$DECL fUniversalTextTrace}
1   60 --
1   61 --  {$IFC NOT WithUObject}
1   62 --      {$DECL fDebugMethods}
1   63 --      {$SETC fDebugMethods := FALSE}          {Must be FALSE}
1   64 --
1   65 --      {$DECL fDbgObject}
1   66 --
1   67 --      {$DECL fTrce}
1   68 --      {$SETC fTrce := FALSE}                  {Must be FALSE}
1   69 --
1   70 --      {$SETC fDbgObject := FALSE}             {Set to FALSE for final libraries}
1   71 --  {$ENDC}
1   72 --
1   73 --
1   74 --  {$SETC fUniversalTextTrace := fTrce}        {Normal}
1   75 --
1   76 --  {$DECL PasteTrace}                          {Generates READLN asking for tracing during Write UT}
1   77 --  {$SETC PasteTrace := FALSE}
1   78 --
1   79 --
1   80 --  TYPE
1   81 --
1   82 --  {$IFC NOT WithUObject}
1   83 --
1   84 --      S255 =  STRING[255];
1   85 --      THeap = Ptr;     {alias for THz}
1   86 --      TClass = Ptr;    {alias for TPSliceTable in UClascal}
1   87 --
1   88 --      TCollecHeader =  RECORD
1   89 --          classPtr:       TClass;
1   90 --          size:           LONGINT;    {number of real elements, not counting the hole}
1   91 --          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data; MAXINT if none alloued}
1   92 --          holeStart:      INTEGER;    {0 = at the beginning, size = at the end; MAXINT = none alloued}
1   93 --          holeSize:       INTEGER;    {measured in MemberBytes units}
1   94 --          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grou the collection by}
1   95 --          END;
1   96 --
1   97 --      TFastString = RECORD            {only access ch[ i] when hole is at end & TString is not subclassed}
1   98 --          header:         TCollecHeader;
1   99 --          ch:             PACKED ARRAY[1..32740] OF CHAR;
1  100 --          END;
1  101 --      TPFastString = ^TFastString;
1  102 --      THFastString = ^TPFastString;
1  103 --
1  104 --
1  105 --      TUTObject = SUBCLASS OF NIL
1  106 --
1  107 --          FUNCTION  {TUTObject.}CREATE(heap: THeap): TUTObject; ABSTRACT;
1  108 --          FUNCTION  {TUTObject.}Heap: THeap;                          {which heap it is in}
1  109 --          PROCEDURE {TUTObject.}FreeObject; DEFAULT;                  {frees just the object, not its }
1  110 --                                                                      {contents}
```

```
1  111 --          PROCEDURE {TUTObject.}Free; DEFAULT;                          {frees the object and its contents}
1  112 --          FUNCTION  {TUTObject.}Class: TClass;
1  113 --          END;
1  114 --
1  115 --      TUTCollection = SUBCLASS OF TUTObject
1  116 --
1  117 --        {Variables}
1  118 --          size:           LONGINT;    {number of real elements, not counting the hole}
1  119 --          dynStart:       INTEGER;    {bytes from the class ptr to the dynamic data}
1  120 --          holeStart:      INTEGER;    {0 means hole at the beginning, size means hole at the end}
1  121 --          holeSize:       INTEGER;    {measured in MemberBytes units}
1  122 --          holeStd:        INTEGER;    {if the holeSize goes to 0, how much to grow the collection by}
1  123 --
1  124 --          FUNCTION  {TCollection.}CREATE(object: TUTObject; heap: THeap; initialSlack: INTEGER): TUTCollection;
1  125 --          FUNCTION  {TCollection.}AddrMember(i: LONGINT): LONGINT;
1  126 --          FUNCTION  {TCollection.}MemberBytes: INTEGER; ABSTRACT;
1  127 --          PROCEDURE {TCollection.}EditAt(atIndex: LONGINT; deltaMembers: INTEGER);
1  128 --          PROCEDURE {TCollection.}InsManyAt(i: LONGINT; otherCollection: TUTCollection; index,
1  129 --                                howMany: LONGINT);
1  130 --          PROCEDURE {TCollection.}ResizeColl(membersPlusHole: INTEGER);
1  131 --          PROCEDURE {TCollection.}ShiftColl(afterSrcIndex, afterDstIndex, howMany: INTEGER);
1  132 --          PROCEDURE {TCollection.}StartEdit(withSlack: INTEGER);
1  133 --          PROCEDURE {TCollection.}StopEdit;
1  134 --          END;
1  135 --
1  136 --      TUTArray = SUBCLASS OF TUTCollection
1  137 --
1  138 --          recordBytes: INTEGER;
1  139 --
1  140 --          FUNCTION  {TArray.}CREATE(object: TUTObject; heap: THeap; initialSlack, bytesPerRecord: INTEGER)
1  141 --                                : TUTArray;
1  142 --          FUNCTION  {TArray.}MemberBytes: INTEGER; OVERRIDE;
1  143 --          FUNCTION  {TArray.}At(i: LONGINT): Ptr; DEFAULT;
1  144 --          PROCEDURE {TArray.}InsAt(i: LONGINT; pRecord: Ptr); DEFAULT;
1  145 --          PROCEDURE {TArray.}InsLast(pRecord: Ptr);
1  146 --          PROCEDURE {TArray.}DelAll;
1  147 --          PROCEDURE {TArray.}DelAt(i: LONGINT); DEFAULT;
1  148 --          PROCEDURE {TArray.}DelManyAt(i, howMany: LONGINT); DEFAULT;
1  149 --          PROCEDURE {TArray.}PutAt(i: LONGINT; pRecord: Ptr);
1  150 --          END;
1  151 --
1  152 --
1  153 --      TUTString = SUBCLASS OF TUTCollection
1  154 --
1  155 --          FUNCTION  {TString.}CREATE(object: TUTObject; heap: THeap; initialSlack: INTEGER): TUTString;
1  156 --          FUNCTION  {TString.}At(i: LONGINT): CHAR;
1  157 --          FUNCTION  {TString.}MemberBytes: INTEGER; OVERRIDE;
1  158 --          PROCEDURE {TString.}ToPAOCAt(i, howMany: LONGINT; pPackedArrayOfCharacter: Ptr);
1  159 --          PROCEDURE {TString.}InsAt(i: LONGINT; character: CHAR);
1  160 --          PROCEDURE {TString.}InsPAOCAt(i: LONGINT; pPackedArrayOfCharacter: Ptr; howMany: LONGINT);
1  161 --          PROCEDURE {TString.}DelAt(i: LONGINT);
1  162 --          PROCEDURE {TString.}DelManyAt(i, howMany: LONGINT);
1  163 --          PROCEDURE {TString.}DelAll;
1  164 --          END;
1  165 --
1  166 --  {$ENDC}
1  167 --
1  168 --      TEnumLevelOfGranularity = (UTCharacters, UTParagraphs);
1  169 --      TLevelOfGranularity = SET OF TEnumLevelOfGranularity;
1  170 --
1  171 --      TCharDescriptor = RECORD      { character descroptor record }
1  172 --              font:           INTEGER;        { font number }
1  173 --              face:           {$IFC LibraryVersion <= 20}TSeteface{$ELSEC}style{$ENDC};          { formating }
1  174 --              superscript:    -128..127;      { number of bits to superscript }
1  175 --              keepOnSamePage: BOOLEAN;
1  176 --              END;
1  177 --
1  178 --      TTabTypes   = (qLeftTab,    qCenterTab,     qRightTab,      qPeriodTab,     qCommaTab);
1  179 --      TTabFill    = (tNoFill,     tDotFill,       tHyphenFill,    tUnderLineFill);
1  180 --      TParaTypes  = (qLeftPara,   qCenterPara,    qRightPara,     qJustPara);
1  181 --
1  182 --      TTabDescriptor = RECORD
1  183 --              position:           INTEGER;            {Location of the tab}
1  184 --              fillBetweenTabs:    TTabFill;           {Fill character for the tab}
1  185 --              tabType:            TTabTypes;          {Type of tab}
1  186 --              END;
1  187 --
1  188 --      TParaDescriptor = RECORD
1  189 --              paragraphStart:     BOOLEAN;  { TRUE if the beginning of the run is also the beginning of a
1  190 --                                              paragraph}
1  191 --  {$IFC WithUObject}
1  192 --              additionalChrInParagraph:   INTEGER;
1  193 --  {$ENDC}
1  194 --              firstLineMargin:    INTEGER;            {Left margin of first line}
1  195 --              bodyMargin:         INTEGER;            {Left margin of subsequent lines}
1  196 --              rightMargin:        INTEGER;            {Right margin}
1  197 --              paraLeading:        INTEGER;            {Paragraph leading}
1  198 --              lineSpacing:        0..63;              {Inter-line spacing }
1  199 --  {$IFC WithUObject}
1  200 --              tabTable:           TArray {OF TTabDescriptor}; { table of tabs  }
1  201 --  {$ELSEC}
1  202 --              tabTable:           TUTArray {OF TTabDescriptor}; { table of tabs  }
1  203 --  {$ENDC}
1  204 --              paraType:           TParaTypes;         {Paragraph adjustment }
1  205 --              hasPicture:         BOOLEAN;            {Is there a picture avaible for this paragraph?}
1  206 --              END;
1  207 --
1  208 --
1  209 --  {$IFC WithUObject}
1  210 --      TTKUnivText = SUBCLASS OF TOBJECT
1  211 --  {$ELSEC}
1  212 --      TUnivText   = SUBCLASS OF TUTObject
1  213 --  {$ENDC}
1  214 --          paragraphDescriptor:    TParaDescriptor;
1  215 --          characterDescriptor:    TCharDescriptor;
1  216 --          maxDataSize:            INTEGER;
1  217 --  {$IFC WithUObject}
1  218 --          data:                   TString;
1  219 --  {$ELSEC}
1  220 --          data:                   TUTString;
```

```
1  221 --   {$ENDC}
1  222 --           itsOurTString:          BOOLEAN;
1  223 --
1  224 --   {$IFC WithUObject}
1  225 --           FUNCTION  {TTKUnivText.}CREATE(object: TObject;
1  226 --                                          itsHeap: THeap;
1  227 --                                          itsTString: TString;
1  228 --                                          itsDataSize: INTEGER) : TTKUnivText;
1  229 --   {$ELSEC}
1  230 --           FUNCTION  {TUnivText.}CREATE(object: TUTObject;
1  231 --                                          itsHeap: THeap;
1  232 --                                          itsTString: TUTString;
1  233 --                                          itsDataSize: INTEGER) : TUnivText;
1  234 --   {$ENDC}
1  235 --           PROCEDURE {TUnivText.}Free; OVERRIDE;
1  236 --           PROCEDURE {TUnivText.}RunToStream;
1  237 --           PROCEDURE {TUnivText.}StreamToRun;
1  238 --           PROCEDURE {TUnivText.}TabTableToArgTbd;
1  239 --           PROCEDURE {TUnivText.}ArgTbdToTabTable;
1  240 --           END;
1  241 --
1  242 --
1  243 --   {$IFC WithUObject}
1  244 --       TTKReadUnivText = SUBCLASS OF TTKUnivText
1  245 --   {$ELSEC}
1  246 --       TReadUnivText   = SUBCLASS OF TUnivText
1  247 --   {$ENDC}
1  248 --
1  249 --   {$IFC WithUObject}
1  250 --           buffer:          TString;
1  251 --   {$ELSEC}
1  252 --           buffer:          TUTString;
1  253 --   {$ENDC}
1  254 --           columnCount:     INTEGER;
1  255 --           dataBeforeTab:   BOOLEAN;
1  256 --
1  257 --   {$IFC WithUObject}
1  258 --           FUNCTION  {TReadUnivText.}CREATE(object: TObject;
1  259 --                                            itsHeap: THeap;
1  260 --                                            itsTString: TString;
1  261 --                                            itsDataSize: INTEGER;
1  262 --                                            LevelOfGranularity: TLevelOfGranularity)
1  263 --                                                                    : TTKReadUnivText;
1  264 --   {$ELSEC}
1  265 --           FUNCTION  {TReadUnivText.}CREATE(object: TUTObject;
1  266 --                                            itsHeap: THeap;
1  267 --                                            itsTString: TUTString;
1  268 --                                            itsDataSize: INTEGER;
1  269 --                                            LevelOfGranularity: TLevelOfGranularity)
1  270 --                                                                    : TReadUnivText;
1  271 --   {$ENDC}
1  272 --
1  273 --           PROCEDURE {TReadUnivText.}Free; OVERRIDE;
1  274 --           PROCEDURE {TReadUnivText.}ReadRun;          { Returns one run of text each time called }
1  275 --           PROCEDURE {TReadUnivText.}Restart;          { Resets the object to read from the begining }
1  276 --
1  277 --           PROCEDURE {TReadUnivText.}ScanTable(VAR rows,
1  278 --                                               tabColumns,
1  279 --                                               tabStopColumns: INTEGER);
1  280 --                                     { Returns number of rows and colums of scrap if a valid table }
1  281 --
1  282 --           FUNCTION  {TReadUnivText.}ReadField(     maxFieldSize: INTEGER;
1  283 --                                              VAR fieldOverflow: BOOLEAN;
1  284 --                                              VAR fieldTerminator: CHAR;
1  285 --                                              VAR tabType: TTabTypes)
1  286 --                                                                  : BOOLEAN;
1  287 --                                                  { Returns one field of text each time called }
1  288 --
1  289 --           FUNCTION  {TReadUnivText.}ReadLine(     maxLineSize: INTEGER;
1  290 --                                             VAR lineOverflow: BOOLEAN;
1  291 --                                             VAR lineTerminator: CHAR)
1  292 --                                                                  : BOOLEAN;
1  293 --                                                  { Returns one line of text each time called }
1  294 --           FUNCTION  {TReadUnivText.}GetParaPicture(heap: THeap)
1  295 --                                                                  : PicHandle;
1  296 --                                               { Copies the picture for the current paragraph into heap }
1  297 --           END;
1  298 --
1  299 --
1  300 --   {$IFC WithUObject}
1  301 --       TTKWriteUnivText = SUBCLASS OF TTKUnivText
1  302 --   {$ELSEC}
1  303 --       TWriteUnivText   = SUBCLASS OF TUnivText
1  304 --   {$ENDC}
1  305 --
1  306 --   {$IFC WithUObject}
1  307 --           FUNCTION  {TWriteUnivText.}CREATE(object: TObject;
1  308 --                                             itsHeap: THeap;
1  309 --                                             itsTString: TString;
1  310 --                                             itsDataSize: INTEGER)
1  311 --                                                                  : TTKWriteUnivText;
1  312 --   {$ELSEC}
1  313 --           FUNCTION  {TWriteUnivText.}CREATE(object: TUTObject;
1  314 --                                             itsHeap: THeap;
1  315 --                                             itsTString: TUTString;
1  316 --                                             itsDataSize: INTEGER)
1  317 --                                                                  : TWriteUnivText;
1  318 --   {$ENDC}
1  319 --           PROCEDURE {TWriteUnivText.}FillParagraph;       {Writes one run of text each time called}
1  320 --           END;
1  321 --
1  322 --   {$IFC NOT WithUObject}
1  323 --       FUNCTION  NewUTObject(heap: THeap; itsClass: TClass): TUTObject;
1  324 --   {$ENDC}
1  325 --
1  326 --   {$IFC fUniversalTextTrace}
1  327 --   VAR
1  328 --       fPrintSecrets:   BOOLEAN;
1  329 --   {$ENDC}
1  330 --
```

```
1  331 --   IMPLEMENTATION
1  332 --
1  333 --   {$IFC fDbgOk}
1  334 --   {$R+}
1  335 --   {$ELSEC}
1  336 --   {$R-}
1  337 --   {$ENDC}
1  338 --
1  339 --   {$IFC fSymOk}
1  340 --   {$D+}
1  341 --   {$ELSEC}
1  342 --   {$D-}
1  343 --   {$ENDC}
1  344 --
1  345 --   {$SETC doTraceUT := FALSE}
1  346 --   {$SetC fTraceUT := doTraceUT AND fUniversalTextTrace}
1  347 --
1  348 --   {$IFC WithUObject}
1  349 --   {$S TKUTMain}
1  350 --   {$ELSEC}
1  351 --   {$S UTMain}
1  352 --   {$ENDC}
1  353 --
1  354 --   {$I libut/UUnivText2.text}
2    1 --
2    2 --
1  355 --
1  356 --   {$IFC WithUObject}
1  357 --   {$S TKUTInit}
1  358 --   {$ELSEC}
1  359 --   {$S UTInit}
1  360 --   {$ENDC}
1  361 --
1  362 --   END.
```

```
1. libtk/uunivtext.TEXT
2. libut/UUnivText2.text

-A-
  additionalChrInP  192•( 1)
  AddrMember        125•( 1)
  ArgTbdToTabTable  239•( 1)
  At                143•( 1)  156•( 1)

-B-
  bodyMargin        195•( 1)
  buffer            250 ( 1)  252•( 1)

-C-
  ch                 99•( 1)
  CHAR               99 ( 1)  156 ( 1)
  characterDescrip  215•( 1)
  Class             112•( 1)
  classPtr           89•( 1)
  columnCount       254•( 1)
  CREATE            107 ( 1)  124•( 1)  140•( 1)  155 ( 1)  225•( 1)  230•( 1)  258•( 1)  265•( 1)  307 ( 1)  313•( 1)

-D-
  data              218•( 1)  220•( 1)
  dataBeforeTab     255•( 1)
  DelAll            146•( 1)  163•( 1)
  DelAt             147•( 1)  161•( 1)
  DelManyAt         148•( 1)  162•( 1)
  dynStart           91•( 1)  119•( 1)

-E-
  EditAt            127•( 1)
  Events             56•( 1)

-F-
  face              173•( 1)
  fillBetweenTabs   184•( 1)
  FillParagraph     319•( 1)
  firstLineMargin   194•( 1)
  font              172•( 1)
  FontMgr            45•( 1)   49•( 1)
  fPrintSecrets     328•( 1)
  Free              111•( 1)  235•( 1)  273•( 1)
  FreeObject        109•( 1)

-G-
  GetParaPicture    294•( 1)

-H-
  hasPicture        205•( 1)
  header             98•( 1)
  Heap              108•( 1)
  holeSize           93•( 1)  121•( 1)
  holeStart          92•( 1)  120•( 1)
  holeStd            94•( 1)  122•( 1)

-I-
  InsAt             144•( 1)  159•( 1)
  InsLast           145•( 1)
  InsManyAt         128•( 1)
  InsPAOCAt         160•( 1)
  INTRINSIC          24•( 1)
  itsOurTString     222•( 1)

-K-
  keepOnSamePage    175•( 1)

-L-
  lineSpacing       198•( 1)

-M-
  maxDataSize       216•( 1)
  MemberBytes       126•( 1)  142•( 1)  157•( 1)

-N-
  NewUTObject       323•( 1)

-P-
  paragraphDescrip  214 ( 1)
  paragraphStart    189•( 1)
  paraLeading       197•( 1)
  paraType          204•( 1)
  PicHandle         295 ( 1)
  PMDecl             52•( 1)
  position          183•( 1)
  PrStdInfo          53•( 1)
  Ptr                85 ( 1)   86 ( 1)  143 ( 1)
  PutAt             149•( 1)

-Q-
  qCenterPara       180•( 1)
  qCenterTab        178•( 1)
  qCommaTab         178•( 1)
  qJustPara         180•( 1)
  qLeftPara         180•( 1)
  qLeftTab          178•( 1)
  qPeriodTab        178•( 1)
  qRightPara        180•( 1)
  qRightTab         178•( 1)
  QuickDrau          46•( 1)   48•( 1)

-R-
  ReadField         282•( 1)
  ReadLine          289•( 1)
  ReadRun           274•( 1)
  recordBytes       138 ( 1)
  ResizeColl        130•( 1)
  Restart           275•( 1)
  rightMargin       196•( 1)
  RunToStream       236•( 1)
```

```
-S-
  S255                84*( 1)
  ScanTable          277*( 1)
  Scrap               57*( 1)
  ShiftColl          131*( 1)
  size                90*( 1)  118 ( 1)
  StartEdit          132*( 1)
  StopEdit           133*( 1)
  Storage             42*( 1)
  StreamToTRun       237*( 1)
  STRING              84 ( 1)
  style              173 ( 1)
  superscript        174*( 1)
  Syscall             51*( 1)

-T-
  tabTable           200*( 1)  202*( 1)
  TabTableToArgTbd   238*( 1)
  tabType            185*( 1)
  TArray             200 ( 1)
  TCharDescriptor    171*( 1)  215 ( 1)
  TClass              86*( 1)   89 ( 1)  112 ( 1)
  TCollecHeader       88*( 1)   98 ( 1)
  tDotFill           179*( 1)
  TEnumLevelOfGran   168*( 1)  169 ( 1)
  TFastString         97*( 1)  101 ( 1)
  THeap               85*( 1)  108 ( 1)
  THFastString       102*( 1)
  tHyphenFill        179*( 1)
  TLevelOfGranular   169*( 1)
  tNoFill            179*( 1)
  TOBJECT            210 ( 1)
  ToPAOCAt           158*( 1)
  TParaDescriptor    188*( 1)  214 ( 1)
  TParaTypes         180*( 1)  204 ( 1)
  TPFastString       101*( 1)  102 ( 1)
  TReadUnivText      246 ( 1)  270 ( 1)
  TSeteface          173 ( 1)
  TString            218 ( 1)  250 ( 1)
  TTabDescriptor     182*( 1)
  TTabFill           179*( 1)  184 ( 1)
  TTabTypes          178*( 1)  185 ( 1)
  TTKReadUnivText    244*( 1)  263 ( 1)
  TTKUnivText        210*( 1)  228 ( 1)  244 ( 1)  301 ( 1)
  TTKWriteUnivText   301*( 1)  311 ( 1)
  tUnderLineFill     179*( 1)
  TUnivText          212 ( 1)  233 ( 1)  246 ( 1)  303 ( 1)
  TUTArray           136*( 1)  141 ( 1)  202 ( 1)
  TUTCollection      115*( 1)  124 ( 1)  136 ( 1)  153 ( 1)
  TUTObject          105*( 1)  107 ( 1)  115 ( 1)  212 ( 1)  323 ( 1)
  TUTString          153*( 1)  155 ( 1)  220 ( 1)  252 ( 1)
  TWriteUnivText     303 ( 1)  317 ( 1)

-U-
  UClascal            41*( 1)
  UnitCS              55*( 1)
  UnitFmt             54*( 1)
  UnitHz              39*( 1)
  UnitStd             38*( 1)
  UObject             35*( 1)
  UTCharacters       168*( 1)
  UTKUniversalText    13*( 1)
  UTParagraphs       168*( 1)
  UUniversalText      15*( 1)

*** End Xref: 135 id's   201 references      [422880 bytes/4864 id's/43132 refs]
```