Dialog Building Block

Toolkit Dialog Building Block

Dialog Boxes

A Dialog Box on the Lisa is a special window which, when displayed, is as wide as the screen, and hangs down right below the menu bar.

Dialog boxes can be used as alternatives to menu commands when an application needs to gather more detail from its user than can be conveniently packaged in a menu.

A ToolKit application can put up a Dialog Box any time its window is active. The Dialog Building Block provides basic structures sufficient to define dialogs, to display them, to specify special behaviour within them, and to interrogate them.

Read No Further

The intended audience for this document is current or prospective direct users of the Dialog Building Block. A basic familiarity with the ideas behind the ToolKit and the CLASCAL language is assumed. ToolKit jargon is unavoidable in a document such as this.

Companion Documentation

The primary documentation for the Dialog Building Block consists of three parts. I list them here in descending order of importance and authority.

[a] The source listing of the INTERFACE of UDialog

[b] The source listings of the ToolKit sample programs, USample and USamDialog.

[c] This document, plus subsequent addenda/errata.

How To Do It

Simple use of the Dialog Building Block involves allocating a TDialogWindow, installing a TDialog, and adding dialog components (dialogImages) to the dialog to define its display and behaviour.

In a typical dialog Box, an Application and its user agree that whatever the user does up in the dialog box is not "for real" until the OK button (or some other action button) is pressed.

Each standard kind of dialog component carries with it some basic assumptions about mouse- and cursor-behaviour. It is

an inherent property of the TCluster component, for example, that one and only one of its checkboxes is selected at any one moment. You do not *program* this behaviour--you select it by the very act of choosing to use a TCluster component.

Thus, an application typically defines the form and behaviour of a dialog box by the simple act of allocating its components. The real action, whereby the Application actually does something to its data structures, is precipitated by the user's pressing a Button in the dialog.

The Application is able to capture control at button-pushing time in either of two ways: [a] By redefining TDialog.ButtonPushed, or [b] by associating a command number with a button, and then fielding that command in the NewCommand methods of its subclasses of either TSelection or TWindow.

Formally, a Dialog is an object which resides in a DialogView, which in turn is installed in some panel of some window. Beginning users may think of the Dialog, the DialogView, and the DialogWindow as confusingly overlapping in function. It is hoped the explanations below, combined with the streamlined functionality of the NewStdDialogWindow procedure, will let the novice user get things done efficiently even before it is obvious what is going on.

Each dialog box you use will involve at least objects of the following classes:

[1] TDialogWindow: its Window. You usually create an *instance* of this class (rather than define a subclass), and this is most expeditiously done by calling the global function NewStdDialogWindow, in which case you also get a panel, a pane, and a dialogView all allocated and properly installed.

[2] TPanel: the Panel in which the dialog takes place; no easy way to subclass this, nor any clear justification for wanting to do so. You get a panel for free if you use NewStdDialogWindow; otherwise, you need to allocate an instance of TPanel yourself, in which case you can have more control over its properties than the NewStdDialogWindow function affords.

[3] TPane: the Pane within the Panel. Allocated automatically when the panel is created.

[4] TDialogView: the view installed in the panel. This is created automatically for you if you use NewStdDialogWindow.

[5] TDialog: A Dialog installed in the dialogView. You

define either an instance of this class or else a subclass of it. More than one Dialog can be installed in the same dialogView (likely to be very useful for programmed instruction and certain other specialized applications, but irrelevant for most users).

Note: TDialogs are the units which will be banked in a Tool Resource File when (if) that module is completed and the Dialog Building Block is integrated with it. Whenever a dialog is allocated, you provide a 4-character "Key" to have permanently associated with the dialog in the Resource File. Unless and until resource files are used, this "Key" plays no real role in anything. Still, it makes sense to assign a unique 4-character key to each different Dialog in your application, so that it will be easier to switch over to Resource Files later.

[6] TDialogImage: the components of the Dialog: checkboxes, buttons, etc. You create these by calling methods of TDialog, such as TDialog.AddStdButton.

Note: DialogImages were, in earlier versions of the Dialog Building Block, called Components, and the terms "Dialog Image", "dialogImage", "dialog Component", and "Component" will be used interchangeably in this document.

Typical use of a dialog box:

-- Call NewStdDialogWindow to define a new dialog box.

If you call the object you have just allocated "myDialogWindow", then the panel and dialogView automatically created for you are as follows:

myDialogWindow.controlPanel is the panel.

myDialogWindow.dialogView is the dialogView installed in the panel (also reachable of course as myDialogWindow.controlPanel.view).

myDialogWindow.mainDialog is initially NIL, but is set to the first TDialog to be installed in the dialogView.

-- Define your Dialog:

(1) Allocate an appropriate TDialog object.

If you are *not* subclassing TDialog, you will allocate an *instance* of TDialog, and at the same time install it in your dialogView with code like:

```
myDialog := myDialogWindow.dialogView.AddNewDialog('XYZ');
```

If you *are* defining your own subclass of TDialog, you will need to install it in the dialogView yourself. This can be done by:

```
dialogView := myDialogWindow.dialogView;
dialogView.AddDialog(TMyDialog.CREATE(...dialogView...));
```

(2) One by one, add the desired components to the Dialog.

If you are *not* subclassing TDialog, then you will do this just after your call to NewStdDialogWindow, probably in your window or selection's NewCommand.

If your *are* subclassing TDialog, then you will do this in your subclass's CREATE method.

DialogImages can be created and added to the dialog in a single step using TDialog methods:

AddStdButton --------	to define a Button
AddStdCluster -------	to define a cluster of checkboxes
AddStdFreeCheckbox ---	to define a free checkbox (one which is not part of a cluster)
AddStdInputFrame ----	to define an Input Frame
AddStdLegend --------	to define a 'Legend' using any Typestyle
AddSysLegend --------	to define a Legend using System Font

Once the dialog structures have been allocated, you can at any point request your main Window to put up the dialog box, with a call to TWindow.PutUpDialogBox. The dialog box will be put up and business proceeds as usual. Events that come in may be dispatched either to the main window or to the dialog box, depending on the settings of various dialogBox parameters.

To take down the dialog box, tell your main window to TakeDownDialogBox. (This can also happen for you automatically if you have said you wish your dialog box to be automatically dismissed under certain circumstances.)

Some thought needs to be given to whether a dialog box will remain allocated even when not in use, or whether it should be freed right away after each dismissal. If you choose the former route (saves time, wastes space, gives sometimes-desirable continuity), you will need to store a reference to the DialogWindow somewhere (probably as a field in your main Window). If you choose to have the dialog structures go away after dismissal, you can get this behaviour by setting the TDialogWindow's field "freeOnDismissal" to TRUE.

Identification of components: IDs and IDNumbers

Some dialogImage types, such as TButton, TCluster, TInputFrame, and , have 'ID's associated with them for identification. DialogImage ids can be either string-valued (e.g. 'Joe') or integer-valued, or both.

String-valued ID's: Only the first 9 characters are significant; all determinations are made in upper-case, with up-shifting handled automatically.

Toolkit Dialog Building Block

Most dialogImages, when created in the 'standard' way, take a character string called itsID as an argument. The upshifted version of the first 9 characters of this ID will form the ID field of the dialogImage FOREVER--no matter how much or how often the text associated with the dialogImage changes.

Using idNumbers (INTEGERS) rather than ID's (STRINGS) is naturally more efficient, and it is anticipated that users will use the string-ID's initially for everything, and will later give idNumbers to any dialogImages they wish to query, for efficiency.

The alternative of using an IDNumber which is also an index into the Phrase File, from whence the text for the object is obtained, (which was the technique used in release TK7D of the ToolKit, before interactive Dialog Layout and Resource Files existed), can still be used, although the process is now more awkward.

For example, if you have a checkbox whose legend you wish to retrieve from the phrase as phrase number xxZnak, then you can have code like:

```
process.GetAlert(xxZnak, thisID);
checkbox := cluster.AddAlignedCheckbox(thisID, FALSE);
checkbox.idNumber := xxZnak;
```

In this example, if you wanted to check whether this particular checkbox is the one currently selected in the cluster, you could find out with code like:

```
IF cluster.hitBox.idNumber = xxZnak THEN ...
```

OR alternatively:

```
CASE cluster.hitBox.idNumber OF
  xxZnak:  ....
```

Constructing a reference from an ID or an IDNumber

Given the ID or IDNumber of a dialog component, you can always obtain a reference to the object itself by using one of the following methods:

```
FUNCTION TDialogImage.ObjWithID(id: S255):
    TDialogImage;

FUNCTION TDialogImage.ObjectWithIDNumber(idNumber:
    INTEGER): TDialogImage;
```

ObjWithID and ObjectWithIDNumber are defined for any subclass of TDialogImage, but can be expected to return NIL unless called from a class which represents a structured object with 'children', such as TDialog (children are its components), or TCluster (children are its checkboxes).

EXAMPLES:

(a) If SELF is a Dialog which has a button whose ID is 'NEXT', then TButton(SELF.ObjWithID('NEXT')) is the button itself.

(b) If SELF is a Dialog which has a cluster whose IDNumber is 429, then TCluster(SELF.ObjectWithIDNumber(429)) is the cluster itself.

(c) If SELF is a cluster which has a checkbox with idNumber 28, then TCheckbox(SELF.ObjectWithIDNumber(29)) is the checkbox itself.

Determining whether a given component has a given ID

If myDialogImage is a some dialog image, one can determine whether it has numeric ID 24 (for example) by just testing whether myDialogImage.idNumber = 24.

To test for a string-valued ID, however, it is better to call the method TDialogImage.HasID, since this will adjust for unequal length strings and for upper-case issues. Thus, to see if myDialogImage has "Jones" as its ID, you can look at the value of myDialogImage.HasID('JONES').

The Standard Dialog Components

TLegend

This type of dialog image consists of one line of text. More than one typeStyle can be used in the line, but typically one deals with a single typeStyle for the entire Legend. Indeed, all the standard ways of creating Legends involve specifying only a single typeStyle; font and face changes are usually brought about through interactive editing.

Legends are used in the Dialog Building Block wherever text needs to be displayed, such as inside Buttons, alongside Checkboxes, as the prompts in Input Frames, and in Legend Headings (the standard kind of heading available through the Dialog Building Block). Legends can also be used directly to display any desired text on the screen.

Creation:

```
FUNCTION TDialog.AddStdLegend(itsID: S255; itsXLoc,
        itsYLoc: LONGINT; itsTypeStyle: TTypeStyle):
        TLegend;
```

"itsID" is a string which will form the characters of the legend; the baseline of the legend will be at (itsXLoc, itsYLoc) and all characters will be in the typestyle given by itsTypeStyle.

FUNCTION TDialog.AddSysLegend(itsID: S255; itsXLoc, itsYLoc: LONGINT): TLegend;

itsID is a string which will form the characters of the legend; the baseline of the legend will be at (itsXLoc, itsYLoc) and all characters will be in system font.

FUNCTION TLegend.CREATE(object: TObject; heap: THeap; itsChars: S255; itsView: TView; itsLocation: LPoint; itsTypeStyle: TTypeStyle): TLegend;

This is the required CREATE method, which you might want to call if you are adding a legend to a dialogImage which is *not* a TDialog.

Other TLegend methods which may be called by applications:

PROCEDURE TLegend.ChangeToPhrase(newPhrase: INTEGER);

Changes the contents of the legend to the string obtained from the phrase file as phrase number newPhrase.

PROCEDURE TLegend.ChangeToString(newString: S255);

Changes the contents of the legend to the string newString.

PROCEDURE TLegend.GetString(VAR itsString: S255);

Retrieves the current contents of the legend.

An aside about System Font:

System Font is used for the menu bar, for window titles on the desktop, and can also be used in Dialogs for display of static text (NOT for user-input of text), such as in Buttons, Checkboxes, etc.

CAUTION: any attempt to send any text in System Font to the dot-matrix printer in anything other than low-res portrait will blow up Lisa Printing. DO NOT DO IT, YOU WILL BE SORRY.

For this and other reasons, we regard the worlds of System Font and of all other fonts as two disjoint kingdoms. Although not all the desired protections may yet be in place, users are cautioned that mixing system and non-system fonts in the same TLegend may cause problems.

Buttons

TButton is the dialogImage subclass which handles all Buttons. A Button is a rectangular shape with rounded corners, which has some text inside it, and is usually used to request some kind of action; common examples are the OK and CANCEL buttons in Lisa dialogs. Other examples are the NEXT button in LisaCalc. As such, the functions of Buttons and of Menus overlap.

You are not expected to subclass TButton, but rather to create objects of type TButton itself; when a button is "pushed" by the user (mouse comes UP inside the button, having gone down either inside the button or somewhere outside other than within editable text), you get control through your TDialog.ButtonPushed method.

Or, if your button has a command number associated with it, the generic TDialog.ButtonPushed will tell your main window to PerformCommand of your main window's NewCommand object for the command number associated with the button that was pushed.

Creating Buttons, and adding them to Dialogs

To add an OK button, you call the TDialog method AddOKButton(noCmdNumber).

To add an OK button, upon the pressing of which you want your main Window's NewCommand to be called with command number 35, you call TDialog.AddOKButton(35).

To add a CANCEL button, you call the TDialog method AddCancelButton(cmdNumber); usually, you will set cmdNumber to noCmdNumber, since you don't want to do anything if the dialog is cancelled.

AddOKButton and AddCancelButton both place their buttons at a likely spot near the right of the dialog box. You can move them anywhere you wish by telling the buttons to offset to the desired location. (Or you can use interactive dialog layout to get them exactly where you want).

If you use both AddOKButton and AddCancelButton, you can be assured that the two buttons will be the same size.

Example: Allocate an OK button, and locate its top-left corner at (480, 30):

```
SELF.AddOKButton(noCmdNumber);
SetLPt(newLocation, 480, 30);
TButton(SELF.ObjWithID(okString)).OffsetTo(newLocation);
```

Buttons other than OK buttons can be created using:

```
FUNCTION TDialog.AddStdButton(itsID: S255; itsXLoc,
    itsYLoc: LONGINT; sameSizedButton: TButton;
    itsCmdNumber: TCmdNumber): TButton;
```

This allocates a standard button, using 'standard' settings for button metrics; it adds the button to the list of components of the dialog, and also returns a reference to the button in case the client might have some use for it.

Example:

```
nextButton := SELF.AddStdButton('Next', 240, 160, NIL, noCmdNumber);
prevButton := SELF.AddStdStdButton('Previous', 240, 200, nextButton, noCmdNumber);
```

This defines a two buttons, one of which says 'Next', is located at (240, 160), and has no command number associated with it. The other button says 'Previous', is the same size as the "Next" button, and is located at (240, 200).

```
FUNCTION TDialog.AddButton(itsID: S255; itsLocation:
    LPoint; itsMetrics: TButtonMetrics;
    sameSizedButton: TButton; itsCmdNumber:
    TCmdNumber): TButton;
```

This allocates a button, using the button metrics supplied as a parameter. If you are dissatisfied with anything about the way AddStdButton does things, you can create your own ButtonMetrics and create your buttons using AddButton rather than AddStdButton.

Anatomy of a Button

A button's properties are governed by

[a] A set of "ButtonMetrics". TButtonMetrics is a record with the following fields:

height:	INTEGER	The height of the button
curvH:	INTEGER	The h-component of the button's roundRect's curvature.
curvV:	INTEGER	The v-component of the button's roundRect's curvature.
typeStyle:	INTEGER	The typestyle to be used for the text inside the button.
expandNum:	INTEGER	Numerator for expansion factor used in computing width (see below)
expandDen:	INTEGER	Denominator for expansion factor used in computing width. (see below)
absMinWidth:	INTEGER	The absolute minimum width acceptable for the button, no matter how short its text.
penState:	PenState	The state the Pen should be in when drawing the button's roundRect.

[b] A chain of "same-sized Buttons"

If more than one button appears in the same dialog, then the user may wish to have a number of them have the same size, for æsthetic reasons. To achieve this, every button has a 'nextSameSizedButton' field, which is another button which will always be kept the same size as it. Any number of buttons can be linked together in a same-sized-button chain.

Essentially, each button has a 'minimum width', which can change as its text changes. It is a function of the width of the text actually in the the button's Legend at the moment, and of the expandNum/expandDen fields of TButton (which, when divided by each other, yield the factor by which the button must exceed its legend in width), and of the relevant buttonMetrics' absMinWidth field.

A global variable, stdButtonMetrics, contains 'standard' settings for button metrics; users may use this record as a point of departure if insisting on departing from the standard.

Button-related methods of TDialogView

NOTE: Since users normally do not subclass TDialogView, the following methods are most often just invoked in their standard ToolKit form rather than redefined in a subclass.

PROCEDURE TDialogView.AbandonThatButton;

This is how to turn off button highlighting, which will still be on at the time ButtonPushed is called. Do NOT try to have done with a pushed button just by telling it to unHighlight, since the dialogView's data structures will not know about it.

PROCEDURE TDialogView.ButtonPushed(button: TButton);

This procedure turns off the button highlighting, takes down the dialog box, and if the button which was pushed has a command-number associated with it, it tells the main window to PerformCommand that command. Most often, this method ends up being called from the client's own TClientDialog.ButtonPushed method by way of a SUPERSELF call, as the last thing done in a dialog. See TDialog.ButtonPushed discussion below.

PROCEDURE TDialogView.PushButton(button: TButton);

This method can be called either automatically by the Dialog Building Block (such as when default dismissal of a dialog box is indicated) or by the application. This procedure passes the request on to the Dialog in

which the button resides.

PROCEDURE TDialogView.SetDefaultButton(button: TButton);

Call this method at dialog creation time to establish which button in a dialog view is to be the 'Default' one. The default button is distinguished on the screen by being drawn with a thicker pen. Default dismissal of a dialog is triggered by a number of conditions, depending on the settings of the response-variables in the dialogBox (dialogWindow). If you set a default button in a dialog for which default dismissal is impossible, it will be a waste of time.

OR:

PROCEDURE TDialog.SetDefaultButton(button: TButton);

This just tells the parent dialogView to set its default button to the indicated button. You can call this or TDialogView's method, to the same effect.

Checkboxes

A checkbox (class TCheckbox) is a dialogImage which consists of a little rectangle which is either selected (filled with black) or not selected (filled with white).

Most uses of checkboxes involve having them grouped together into a cluster--see below. But checkboxes can also be used on their own, so-called 'Free Checkboxes', and to create these, you can call:

FUNCTION TDialog.AddStdFreeCheckbox(itsID: S255; itsXLoc, itsYLoc: LONGINT): TCheckbox

Example: checkbox := SELF.AddStdFreeCheckbox('Fried Fish', 250, 40);

This installs a free checkbox into a dialog, with the legend 'Fried Fish' and located at (250, 40). The actual box dimensions, and the space between the box and its legend, and the choice of font for the legend, are all standard.

FUNCTION TDialog.AddFreeCheckbox(itsID: S255; itsXLoc, itsYLoc: LONGINT; boxWidth: INTEGER; boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox

This is the most general way to add a free checkbox, allowing you complete control over all the display parameters.

Example I:

```
MakeTypeStyle(fatModern, size12Pitch, [bold], myTypeStyle);
SetPt(myLabelOffset, 10, -3);
checkbox := SELF.AddFreeCheckbox('Fried Fish', 250, 40, 20, 11, TRUE, myLabelOffset,
  myTypeStyle);
```

This installs a free checkbox into a dialog, with the legend 'Fried Fish' and located at (250, 40). The checkbox itself will be 20 x 11 pixels, and the legend's baseline will be located 10 pixels to the right of the box and 3 pixels above the botton of the box; the legend will be in 12-pitch modern type, bold face.

Example II:

```
MakeTypeStyle(famModern, size12Pitch, [bold], myTypeStyle);
checkbox := SELF.AddFreeCheckbox(noID, 400, 120, 25, 17, FALSE, zeroPt, myTypeStyle);
```

This installs a large free-checkbox into a dialog, with NO legend. Its top-left corner is located at (400, 120), and the box itself is 25 x 17 pixels.

FUNCTION TDialog.AddBigFreeCheckbox(itsID: S255; itsXLoc, itsYLoc: LONGINT): TCheckbox

This is like AddStdFreeCheckbox, except it defines a much bigger box, with a larger font used for the legend.

To determine whether a given free checkbox is currently selected (ON; filled with black) or not (OFF; filled with white), you look at its isSelected field.

Example III:

To determine whether the checkbox whose ID is 'Extra Milk' is on or not,

```
IF TCheckbox(SELF.ObjWithID('EXTRA MILK')).isSelected THEN
   {it's on -- do what needs to be done}
ELSE
   {it's off -- act accordingly}
```

Clusters

A cluster (class TCluster) is a dialogImage which contains a list of checkboxes. In a cluster, one and only one of its checkboxes is selected at any given moment. When a cluster is told to select a particular checkbox, it first deselects its currently-selected checkbox.

TCluster has a field hiLitBox which indicates which of the boxes is currently selected (highlighted). To find the checkbox which is currently selected in myCluster, look at myCluster.hiLitBox; to determine action based on which checkbox is chosen in a cluster, you can CASE on myCluster.hiLitBox.idNumber if you use ID Numbers. If you use string-valued ID's only, then you can have code of the form:

```
box := myCluster.hiLitBox;
IF box.HasID('ORANGES') THEN ...
ELSE
IF box.HasID('Lemons') THEN ...
etc...
```

Toolkit Dialog Building Block

To create a typical cluster, use the TDialog method AddStdCluster; this creates the cluster, but it is at the moment empty. Now give it its desired checkboxes, by using any of the following methods:

```
FUNCTION TCluster.AddAlignedCheckbox(itsID: S255;
      selectThisOne:    BOOLEAN): TCheckbox
```

This is the easiest way to add a checkbox to a cluster; you give the text to be shown as the first argument, and for the second one you specify whether this should be the box initially selected in the cluster.

Example:

```
cluster := SELF.AddStdCluster('Fruit', 100, 200);
cluster.AddAlignedCheckbox('Oranges', TRUE);
cluster.AddAlignedCheckbox('Lemons', FALSE);
```

This adds a cluster which has two checkboxes, reading 'Oranges' and 'Lemons', to the dialog, with the Oranges box being selected. The cluster's location (100, 200) becomes the location of the first checkbox; the second checkbox is placed to the right of the first, at a standard distance.

(note the ID of the cluster, 'Fruit', will not show up in the dialog, but it is a name by which the cluster is known for the purposes of identification [e.g., if one did not store a reference to it, one could determine it by SELF.ObjWithID('Fruit')], and will appear as the cluster's title during interactive layout.)

CAUTION: AddAlignedCheckbox, at the moment, keeps adding off to the right, taking no account of the right edge of the screen. This could change later, if demand is heard. See the example on the next page under TDialog.AddRowOfboxes for an easy way to circumvent the problem by simply individually positioning the first checkbox of each new row.

```
FUNCTION TCluster.AddCheckbox(itsID: S255; itsLocation:
   LPoint; boxWidth: INTEGER; boxHeight: INTEGER;
   wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle:
   TTypeStyle; selectThisOne: BOOLEAN): TCheckbox
```

This is the most general way to add a checkbox to a cluster, allowing you complete control over all the display parameters.

```
PROCEDURE TCluster.AddRowOfBoxes(numberOfBoxes: INTEGER;
   startingIDNumber: INTEGER; boxWidth: INTGER; boxHeight:
   INTEGER; boxSpacing: INTEGER);
```

This allows you to add a whole row of checkboxes in one step.

Example:

```
cluster.AddRowOfBoxes(12, 201, 20, 14, 25);
```

This adds 12 checkboxes to the cluster. They are given ID numbers 201 through 212; each box is 20 x 14 in size, and boxes are 25 pixels apart.

```
FUNCTION TDialog.AddRowOfBoxes(itsID: S255; itsXLoc,
   itsYLoc: LONGINT; numberOfBoxes: INTEGER;
   startingIDNumber: INTEGER; boxWidth: INTGER; boxHeight:
   INTEGER; boxSpacing: INTEGER): TCluster;
```

This TDialog method allows you to allocate a cluster and stuff it full of a row of (unlabeled) checkboxes in one step (note that by comparison, to use TCluster.AddRowOfBoxes, you need first to have defined the cluster)

Example: (SELF is a TDialog here)

```
cluster := SELF.AddRowOfBoxes('Vibble', 20, 30, 6, 101, stdBoxWidth, stdBoxHeight,
     stdBoxSpacing);
SetLPt(secondRow, 20, 50);
checkbox := cluster.AddCheckbox(noID, secondRow, stdBoxWidth, stdBoxHeight,
     stdBoxSpacing, FALSE, zeroLPt, sysTypeStyle, FALSE);
checkbox.idNumber := 107
cluster.AddRowOfBoxes(5, 108, stdBoxWidth, stdBoxHeight, stdBoxSpacing);
```

This adds 2 rows of 6 checkboxes each to the cluster. They are given ID numbers 101 through 112. The first checkbox of the second row has to be defined in greater detail so that its new location can be specified.

Input Frames

An Input Frame is a dialogImage which allows the application to obtain character input from its user. Its basic elements are:

[a] A Prompt -- this is a TLegend, and forms the request for input (e.g. "Please Type Your Name Here:"); this involves a location, a typestyle, and the actual text to be shown.

[b] Input specifications: Where the input should show up on the screen; what typestyle should be used for echoing characters typed; and how many characters can be accepted.

[c] Borders -- how far beyond the prompt and input areas should the catchement area for mousepresses be? (in essence, this yields a hitRect).

The "Standard" input frame uses system font for its prompt, 12-pitch modern type for its input font, and has a standard Borders setting, which is illustrated in the sample programs.

Important methods related to Input Frames:

```
FUNCTION TDialog.AddStdInputFrame(itsID: S255;
   itsXLoc, itsYLoc: LONGINT; maxInputChars:
   INTEGER): TInputFrame;
```

Use this method to create a Standard input frame, with prompt in System Font, input characters echoed in

12-pitch Modern. The location you provide is for the baseline of the prompt; the location of the input area is automatically determined.

```
FUNCTION TDialog.AddInputFrame(itsID: S255;
   promptLocation: LPoint; promptTypeStyle:
   TTypeStyle; inputLocation:LPoint;
   inputTypeStyle: TTypeStyle; maxInputChars:
   INTEGER; itsBorders: Rect; drawInputLRect:
   BOOLEAN; drawHitLRect: BOOLEAN): TInputFrame;
```

Call this method if AddStdInputFrame does not serve your purposes.

```
PROCEDURE TDialog.SelectInputFrame(inputFrame:
    TInputFrame);
```

This method, when called, replaces the selection associated with the dialog's panel with a 'Select All' on the user-input contents of the input frame.

```
PROCEDURE TInputFrame.SupplantContents(newContents:
    S255);
```

Call this method to replace the 'user-input' in the input frame with the specified string.

```
PROCEDURE TInputFrame.GetContents(VAR contents: S255);
```

Call this method to find out what the current user-input is.

Interactive Dialog Layout

Whenever a dialog box is up, if you are running a version of software built with debugging turned on, you can select 'Edit Dialog' from the DEBUG menu, and the current version of your dialog box will temporarily be replaced by an editable counterpart.

During interactive layout, you have the opportunity to change two things:

[a] Locations of things -- do by grabbing the tabs provided and dragging them to where you want them to be, then letting up. Use UNDO to reverse the operation.

[b] The textual content of any TLegend object, whether in a button, a checkbox, an inputFrame, or by itself.

Layout is accomplished through the use of 'Layout

Boxes', which are little boxes with little title tabs. DialogImages which have ID's display those ID's in the title tab; dialogImages which do not themselves have ID's (such as TLegends) have a smaller title tab with no text in it.

It is important to realize that what shows up in the title table is the ID of the dialogImage *as it is defined in your source program.*

HOW TO STOP EDITING A DIALOG -- VERY IMPORTANT

The only way to stop editing a dialog is to select 'Stop Editing Dialog' from the DEBUG menu.

WHY DO INTERACTIVE EDITING?

The idea is that the results of the editing are stashed in a Resource File. Subsequently, when you wish to create another dialog of the same sort, you get it from the resource file rather than defining it with TDialog.CREATE--but all of the methods of TDialog that you have redefined or accepted will apply as usual. This mechanism has however not been completed, and until it is, interactive Dialog Editing is of limited use.

If you have a dialog of the sort which is freed and the reallocated each time it is needed, then interactive dialog editing without resource files is nonsense for you. If you keep a dialog around once it is allocated, then even in the absence of resource files, you can have the results of your interactive editing saved by making a Stationery pad of the document you were working on when you did the editing; then all documents built from that Stationery pad will show the edited version of the dialog.

Multiple Dialogs in a DialogView

A dialogView can have any number of Dialogs installed in it, any of which can be either active (visible and playing a role in both output and input) or inactive (not displayed and not fielding input, but fully allocated and waiting in the wings).

We have already seen how an instance of TDialog can be added to a dialogView using TDialogView.AddNewDialog, and how an already-allocated object can be added to a dialogView using TDialogView.AddDialog.

In addition, the following three methods of TDialogView are of interest to users wishing to use multiple dialogs in the same dialog view:

```
PROCEDURE {TDialogView.}ActivateDialog(dialog: TDialog; whichWay: BOOLEAN);
PROCEDURE {TDialogView.}RemoveDialog(dialog: TDialog; andFree: BOOLEAN);
PROCEDURE {TDialogView.}ReplaceDialog(oldDialog, newDialog: TDialog);
```

Miscellaneous notes, in no particular order

[1] Sometimes the application wishes to gain control at the moment a checkbox is hit, so that (for example) it can change the display. To do this, it should implement TDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN). The default CheckboxHit method of TDialog does nothing other than pass the message up the line to its DialogView, whose default CheckboxHit method does nothing.

Your own implementation of TMyDialog.CheckboxHit will presumably determine which checkbox was hit, and which cluster that checkbox was in, if any, and take appropriate action to modify the display or whatever. The "Print Manager" sample dialog in USamDialog gives an example.

[2] The global procedure NewStdDialogWindow gives you a dialogWindow with one panel. To get a multiple-paneled dialog box, you can either call NewStdDialogWindow, and then tell that one panel to divide into two by using TPanel.Divide (this should work fine, provided that the properties of the first panel created automatically suit the requirements of at least one of your target panels), or you can call TDialogWindow.CREATE in the first place, in which case you will need to create all your panels explicitly (the first with TPanel.CREATE and the rest with TPanel.Divide). The "Headings and Margins..." dialog brought up from the Page Layout menu gives an example.

[3] Subclassing TDialogView -- this is possible, and indeed is illustrated in the "Demo Dialog" in the sample program USamDialog. In order to allow dialog-like behaviour to coexist with non-dialog behaviour, a special set of TDialogView methods, all with names starting with "X", are available for redefinition in your subclass of TDialogView.

These methods are: XDraw, XCursorAt, XMousePress, XMouseMove, and XMouseRelease. The default implementation of all of these methods is empty.

XDraw is called from TDialogView.Draw; the idea is that all the "dialog" parts of your dialogView are drawn automatically, and then your XDraw is called in case you wish to add something else to the display.

The other 4 methods, relating to mousing, are all 'escapes', called only when no dialog component lays

claim to the mouse. Thus, if the mouse goes down in a DialogView, first all the components of all of its active Dialogs are given a chance to claim the mouse; if none does, then XMousePress is called to give the non-dialog portion of your DialogView a chance to claim the mouse.

[4] DialogView can be shown in an application's main window as well as up in a dialog box. The application in this case needs to do a bit of extra work to assure that the "Edit Dialog" menu command in the DEBUG menu is suitably enabled and respected. The sample program USamDialog illustrates this use.

[5] The dialog component "TPicObject" is implemented but totally untested and not usable without some effort and risk. The idea of this component is that it can be filled with a QuickDraw picture, by using PASTE at some suitable moment when some suitable thing is selected. This remains a good idea which can't really be used on the first release of the ToolKit.

[6] It is possible to use dialog boxes on the ToolKit without using the Dialog Building Block. Class TDialogBox is defined in the Generic Application ("ABC's"), but serves largely as a front-end to the Dialog Building Block; anyone wanting to use dialogs but not wanting to use the ToolKit's Dialog Building Block could subclass TDialogBox in some other way. This would involve considerable reinvention of the wheel however.

[7] It is possible to use the Dialog Building Block to display a dialog in a panel of an application's main window rather than up in a dialog box. The sample program 'SamDialog' illustrates this.

[8] Each Dialog View operates in one of two ways when it comes to mousing among free checkboxes (i.e., checkboxes which are not members of a cluster). The default is that each time the mouse, while down, is dragged through the domain of a free checkbox, that checkbox is toggled. Some users may want to have a different kind of behaviour, whereby once the mouse has toggled one free checkbox OFF (for example), it will only ever be able to turn other ON boxes OFF -- i.e., it becomes a paintbrush which only paints boxes OFF. Of course, corresponding behaviour will happen if the first free checkbox the mouse encounters once down is one which is currently OFF; in this case, the mouse would become an instrument for painting OFF boxes ON, and doing nothing with already-ON boxes. You can obtain this alternative behaviour by setting the TDialogView field "paintFreeBoxes" to TRUE.

[9] Be sure to get the latest errata sheet for this document from Barry Haynes.

To Lisa ToolKit users

From:

Subject: Dialog Building Block Documentation: Addenda and Corrigenda

Date: 9 April 1984

Translatable Dialogs

When it recently became known that Resource Files would not be a part of the Spring Release of the Lisa ToolKit, the Dialog Building Block, having based its design on the assumption that the results of interactive dialog layout would be savable in a Resource File, found itself in the awkward position of lacking any convenient way for text within dialogs to be translated into other languages.

Since ToolKit interfaces have now been frozen, all facilities described in the document "ToolKit Dialog Building Block" dated 27 March 1984 have to be retained intact, so a way out of the no-Resource-File imbroglio has been found by adding several new Methods to the Interface, while retaining the full functionality of all methods described in the 27 March document. These new methods basically go back to the TK7D design of getting text for dialog components from a Phrase File, but add the ability to specify locations as well as textual content in the Phrase File.

No changes have been made to the interface of any existing methods, and existing code calling the Dialog Building Block should continue to work as is.

To use the new phrase-file-based methods, you place entries in the Phrase file using the following syntax:

```
<phrase number>
<text> @ <h-coordinate> , <v-coordinate>
```

For example, consider the following entry in a Phrase File

```
228
Next Question @ 140, 220
```

This associates the IDNumber 228 with the text "Next Question" and with the location (140, 220). The "@"-sign is the delimiter signalling the end of the text and the beginning of the location. The two locations must be separated by a comma. Spaces are optional before and between.

If in an application's source program the identifier phNextQuestion is declared to be a CONST with value 228, then the call:

```
button := dialog.NewButton(phNextQuestion, stdButtonMetrics, prevButton,
                           uNextQuestion);
```

will add a button to a dialog, putting the text "Next Question" inside the button. Standard button metrics will be used, and the size of this new button will be synchronized with the size of a previously created button called "prevButton"; the top-center point of this new button will be located at (140, 220) in its view. The

button will have phNextQuestion (228) as its idNumber, and when it is pushed by the user, the main window's NewCommand will be called with uNextQuestion as the command-number argument.

In similar ways, all the other standard dialog components can now be created such that they get their text and locations from a phrase file, and use the index into the phrase file as the IDNumber for the component. Where necessary, some methods get only the text (such as TCluster.NewAlignedCheckbox), or only the location (such as TDialog.NewCluster).

The complete list of new methods is as follows:

(A) New Methods of TDialog

```
FUNCTION TDialog.NewButton(itsPhrase: INTEGER; itsMetrics: TButtonMetrics;
                    sameSizedButton: TButton; itsCmdNumber: TCmdNumber):
                    TButton;
```

Text and locations obtained from phrase file; use stdButtonMetrics for itsMetrics to get standard button metrics.

```
FUNCTION TDialog.NewCluster(itsPhrase: INTEGER): TCluster;
```

Location obtained from phrase file. No text involved

```
FUNCTION TDialog.NewFreeCheckbox(itsPhrase: INTEGER; boxWidth: INTEGER;
                    boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point;
                    itsTypeStyle: TTypeStyle): TCheckbox;
```

Text and location obtained from phrase file.

```
FUNCTION TDialog.NewInputFrame(itsPhrase: INTEGER; promptTypeStyle: TTypeStyle;
                    inputOffset: Point; inputTypeStyle: TTypeStyle;
                    maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect:
                    BOOLEAN; drawHitLRect: BOOLEAN): TInputFrame;
```

Text and location for the prompt obtained from phrase-file; "inputOffset" parameter tells how far to move from the end of the prompt to find the location of the "input" rectangle. Locations of both the prompt and the input rectangle indicate where the *Baseline* of the first character is or would be.

```
FUNCTION TDialog.NewLegend(itsPhrase: INTEGER; itsTypeStyle: TTypeStyle):
                    TLegend;
```

Text and location obtained from phrase-file entry. Use sysTypeStyle for itsTypeStyle to get the legend in System Font. Otherwise, use global procedure MakeTypeStyle (defied in UDraw) to construct a typeStyle to pass to this method, or to any method which calls for a TTypeStyle parameter).

```
FUNCTION TDialog.NewRowOfBoxes(itsPhrase: INTEGER; numberOfBoxes: INTEGER;
                    startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight:
                    INTEGER; boxSpacing: INTEGER): TCluster;
```

Location of the first box obtained from phrase file. No text involved.

(B) New Methods of TCluster

```
FUNCTION TCluster.NewAlignedCheckbox(itsPhrase: INTEGER; selectThisOne:
                    BOOLEAN): TCheckbox;

FUNCTION TCluster.NewCheckbox(itsPhrase: INTEGER; boxWidth: INTEGER; boxHeight:
                    INTEGER; wantLabel: BOOLEAN; labelOffset: Point;
                    itsTypeStyle: TTypeStyle; selectThisOne: BOOLEAN):
                    TCheckbox;
```

While it is true, as stated above, that no changes are necessary to any exising code, all new users are encouraged to use these new methods wherever they are appropriate. Existing users who wish their applications to be portable overseas with ease are encouraged to follow suit as time permits.

So What Use is Interactive Dialog Layout now?

Not very much, unless/until Resource files are incorporated. Sadly, the prospect of use of Resource Files meant that one could be rather extravagent in several aspects of internal design. Now that Resource files will not be with us, we are left with the price of the design without the benefit. I am very sorry about it, and I hope that the inefficiencies in space and time will not be that grievous.

But interactive dialog layout still can be used in the following two ways:

[a] If you wish to use a non-blank Stationery Pad as the point of departure for your application, then any edits to any dialogs that were prevalent at the time the model document was made into a Stationery Pad will be incorporated in that Stationery.

But note that this only applies to dialogs that are allocated at start-up time and then never deallocated. Dialogs which are freed and then allocated afresh when needed will always come up as specified in the Source Program/Phrase File.

[b] To get a dialog to be exactly as you want, you can use Interactive Layout, and then use the ToolKit Debugger to inspect the locations of the components. You can then enter those precise coordinates into the phrase file. This is a pretty lo-tech use of the feature, but it beats dozens of iterations between development system and office system.

Further Notes

(1) On page 18 if the 27 March document, additional notes (4) and (7) are essentially identical, giving you some idea of the time of night when the memo was written. Strike either entry.

(2) The 27 March document neglected to mention an important method of TDialog, namely TDialog.PrepareToAppear. Whenever a dialog window is about to be put up, all dialogs in all panels of the window which have TDialogView type view installed in them are sent the message to PrepareToAppear. This can be useful if a dialog which is kept around when not in use must be reformatted or somehow adjusted for the particular circumstance at hand before it is redisplayed. Calls to methods like TInputFrame.SupplantContents and TDialog.SelectInputFrame are anticipated in clients' redefinitions of PrepareToAppear.

==========

From:

To: Whoever still cares

Subject: The built-in headings and margins facility in the ToolKit

Date: 24 April 1984

==========

The ToolKit's standard Headings and Margins... dialog provides a uniform interface for creating and editing headers and footers and for specifying page margins.

This dialog has fallen rather short of its original goals, in the following three ways:

(1) Margins are set using a primitive checkbox dialog rather than by dragging around some nice margin-setting icons.

(2) The only kinds of headings supported are text headings. Arbitrary graphical headings, created by pasting from Universal Graph, haven't been implemeneted yet.

(3) Some awkward limitations are still present in the user interface. These do not limit what can be achieved, but do make some things harder to do. These are all discussed below.

Thus, we have a general and workable facility that didn't have time to get polished. Its quite general functionality is unfortunately packaged in a not so insanely great costume.

With these apologies out of the way, I offer now a ---

Brief Users' Guide:

When you request Headings and Margins... from the Page Layout menu, a dialog box appears with an upper (status) panel and a lower (layout) panel. The layout panel shows an actual-size image of a prototype page of the printer for which the document is formatted. The clear area here is the 'margins' area, in which the user is most likely to wish to locate his headings. The shaded area is the 'body' area, within which the contents of the associated view will be printed.

All headings defined for the view are shown in the layout panel. You can move any heading anywhere you wish on the page, by grabbing its title tab with the mouse, moving the heading where you wish it to be, and then letting go.

You can edit the text of any heading in the usual way.

The following two variables are predefined for headings: {PAGE} indicates that the page number should be substituted, and {TITLE} asks for the window title. Thus, a heading which in the Layout Panel reads

<<< this is page {PAGE} of the window named {TITLE}!>>>

will show up, when printed, as something like:

<<< this is page 23 of the window named Annual Report!>>>

How to Create a New Heading

First choose the properties the new heading is to have, up in the status panel; then press Launch New Heading. A new heading, with text ' --- New Heading --- ' is created, and is located in the layout panel somewhere near the edge of the page that corresponds to the page alignment chosen.

The new heading appears with all its text selected, so that you can immediately type in the actual text you wish to have for your heading. Do text editing and typeStyle specifications in the usual fashion.

Finally, locate the heading exactly where you wish on the page, by grabbing its title tab and dragging it.

How to Delete an unwanted Heading

Select its title tab, and then request CLEAR. This action is not at present undoable.

How to Specify Page Margins

Check inches or centimeters down in the lower part of the status panel, and then click in the checkboxes until they reflect the margins you want. Then push the Install Margins button.

Undo:

You can UNDO any text edits, as well as any positioning of a box. You can NOT, in this version, UNDO the installation of margins or the launch of a new Heading.

Major Cautions:

(1) The margins-specification checkboxes do NOT necessarily reflect what the current settings for the margins are. The way to ensure that a particular margins specification is used is to click in the appropriate boxes so that the checkboxes selected reflect the desired margins, and then press the button entitled Install Margins.

(2) Some of a heading's properties can only be specified at creation time. You can change the text of any heading any time, as well as its type style and its location. But the easiest way to change its page-alignment, or the specifications of which page(s) it is to appear on (odd only, even only, or both odd and even; the minimum and maximum applicable page numbers) is to:

[a] Select the text you wish to retain from the heading, and copy it top the Clipboard.

[b] Throw away the old unwanted heading (using CLEAR after selecting the title tab of the heading)

[c] Specify the desired settings for the heading in the status panel, then press button "Launch New Heading" to get a new heading created with the properties you want. The new heading will appear with all its

default text selected.

[d] Finally, request PASTE to overlay the default text with the text from your earlier heading.

(3) The typestyles used in the Layout Panel for the variables {PAGE} and {TITLE} are irrelevant at printing time; when a variable is substituted for, the typestyle used for the print-time text is the typestyle associated with the last character *before* the variable. It is ok for that character to be a space, and indeed the new headings generated by the Launch New Heading button have just such a leading space for just this purpose.

Notes:

(1) The headings you see in the layout panel may or not be identical to the headings which will actually be printed, depending on whether they do not or do invoke the variables {PAGE} and {TITLE}. If in fact there is one or more such variables in a heading, then at print time, the actual bounds of the heading may be different. What you are assured is that the actual heading printed will be located on the printed page according to the same rules that locate the editable version in the layout panel.

Hence, for example, if you have a heading which has center justification, and you have its center located at the top center of the page, then you can be assured that the actual heading to be printed will also have its center located at the top center of the printed page, no matter how wide it is. Similar remarks hold for left and right justification.

(2) All headings specified, whatever their circumstances, show up in the layout panel. You may well have a number of headings occupying the same place, such as if you want one kind of heading on odd-numbered pages and another on even-numbered pages, but both of them occupying the same relative location on their respective pages. These headings, when stacked up one upon the other in the layout panel, may be difficult to read, and you will probably want to move them apart while you do text editing, then stack them back up again.

(3) You can always determine which headings will actually be printed, where on the page, on which pages, by entering page-preview mode in the relevant panel.

(4) But while the Headings and Margins... dialog is up, don't expect any portion of your main window seen peeking out below the dialog box to reflect the currently-in-flux situation; while the dialog box is up, page-preview mode for the panel in question will show the source versions of the headings, without variable substitution.

```
1     1 --  (*        >>>>>>>>  U D I A L O G  <<<<<<<<
1     2 --
1     3 --  *)
1     4 --
1     5 --  {$SETC forOS := TRUE}
1     6 --
1     7 --  UNIT UDialog;  {Copyright 1984 by Apple Computer, Inc}
1     8 --
1     9 --  {04/25/84 0015  Added field TEditLegendSelection.tripleClick, and methods TEditLegendSelection.
1    10 --                        MousePress, MouseMove, and MouseRelease}
1    11 --  {04/23/84 1210  Removed all references to 'underEdit' field of TDialogImage}
1    12 --
1    13 --  {$Setc IsIntrinsic := TRUE }
1    14 --
1    15 --  {$IFC IsIntrinsic}
1    16 --  INTRINSIC;
1    17 --  {$ENDC}
1    18 --
1    19 --
1    20 --  INTERFACE
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 --
1    30 --    {$U libtk/UABC}             UABC,
1    31 --    {$U libtk/UUnivText}        UTKUniversalText,
1    32 --    {$U libtk/UText}            UText;
1    33 --
1    34 --
1    35 --  CONST
1    36 --      UDialogVersion  = 'UDialog 25Apr84 16:30';
1    37 --
1    38 --  (*
1    39 --                              -----  Dialog Building Block for the ToolKit ------
1    40 --
1    41 --
1    42 --
1    43 --   The Dialog Building Block provides the following standard kinds of dialog images:
1    44 --
1    45 --         Button            A Lisa-style button (a round-cornered Rectangle for pushing, with text inside it)
1    46 --         Checkbox          A checkbox (a box for checking, plus an optional associated textual label)
1    47 --         Cluster           A set of related checkboxes of which only one is selected at a time
1    48 --         InputFrame        A place for keyboard input to be inhaled
1    49 --         Legend            A character string, together with font & face information
1    50 --
1    51 --         TextDialogImage   A box of text managed by the Text editor  (largely untested)
1    52 --         PicObject         A QuickDraw picture  (never tested; probably not bankable; status uncertain)
1    53 --
1    54 --   The basic bankable dialog entity which can be stashed into/retrieved from a Resource File
1    55 --   is the class TDialog.  For each different kind of dialog box you want, you will typically define
1    56 --   another subclass of TDialog.
1    57 --
1    58 --   To EDIT a dialog interactively, you must:
1    59 --        (1)  Have the menu items 'Edit Dialog' and 'Stop Editing Dialog' in your phrase-file
1    60 --        (2)  If the dialog is viewed in your main window rather than in a dialog box, (such as Preferences)
1    61 --               then your own main Window.CanDoCmd should enable uEditDialog whenever the dialog to be editted
1    62 --               is unambiguously selected in the window and there is not a dialog box up;  in this
1    63 --               case,  the dialog editting takes place in a dialog box whereas the dialog itself resides
1    64 --               in the main window.
1    65 --
1    66 --        CAUTION:  Until Resource Files are incoporated, the edits to a dialog are local to the document
1    67 --                  in which you made the edits, as well as documents made from a stationery pad made from
1    68 --                  that document.
1    69 --
1    70 --     How to have your own view be a subclass of TDialogView, and still do all of its normal View things.
1    71 --     while having the Dialog Building Block handle everything that occurs which is relevant to
1    72 --     its dialogs:
1    73 --
1    74 --      (a) To draw the non-dialog parts of the view,  implement method TDialogView.XDraw
1    75 --      (b) To set the cursor in the non-dialog parts of the view,  implement method TDialogView.XCursorAt
1    76 --      (c) Implement XMousePress, XMouseMove, and XMouseRelease instead of their non-x counterparts
1    77 --
1    78 --  *)
1    79 --
1    80 --  TYPE
1    81 --
1    82 --  S4 = STRING[4];
1    83 --
1    84 --  TId = STRING[IDLength];
1    85 --
1    86 --  TButtonMetrics =
1    87 --      RECORD
1    88 --          height:           INTEGER;
1    89 --          curvH:            INTEGER;
1    90 --          curvV:            INTEGER;
1    91 --
1    92 --          typeStyle:        TTypeStyle;
1    93 --
1    94 --          expandNum:        INTEGER;    {a button's min width is its text's with times this numerator}
1    95 --          expandDen:        INTEGER;    {   ... divided by this denominator}
1    96 --
1    97 --          absMinWidth:      INTEGER;
1    98 --          penState:         PenState;   {for drawing the round-rect}
1    99 --      END;
1   100 --
1   101 --  TStringKey =     RECORD          {Keys for Dialogs in Resource Files}
1   102 --                       trueKey:     LONGINT;
1   103 --                       key:         S4;
1   104 --                   END;
1   105 --
1   106 --
1   107 --
1   108 --  {-------------------------------------------------------------------------------------------------------}
1   109 --
1   110 --                                      { ********* CLASSES ********* }
```

```
1  111 --
1  112 --   { ------------------------------ classes implemented in file UDialog2 ------------------------------ }
1  113 --
1  114 --
1  115 --   TDialogWindow = SUBCLASS of TDialogBox  {which itself is in UABC}
1  116 --
1  117 --       controlPanel:    TPanel;        {One with a dialogView in it; may be told to push its dflt button}
1  118 --       dialogView:      TDialogView;   {the view installed in SELF.controlPanel}
1  119 --       mainDialog:      TDialog;       {the first dialog installed in SELF.dialogView}
1  120 --
1  121 --     {Creation/Destruction}
1  122 --       FUNCTION TDialogWindow.CREATE(object: TObject; heap: THeap; itsResizability: BOOLEAN;
1  123 --           itsHeight: INTEGER; itsKeyResponse, itsMenuResponse, itsDownInMainWindowResponse: TDiResponse)
1  124 --                                  : TDialogWindow;
1  125 --
1  126 --     {Showing and Hiding}
1  127 --       PROCEDURE TDialogWindow.Appear;  OVERRIDE;
1  128 --       PROCEDURE TDialogWindow.BeDismissed; OVERRIDE;
1  129 --       FUNCTION  TDialogWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;  OVERRIDE;
1  130 --       PROCEDURE TDialogWindow.Disappear; OVERRIDE;
1  131 --
1  132 --     {Commands}
1  133 --       FUNCTION  TDialogWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  134 --
1  135 --     END;  {TDialogWindow interface}
1  136 --
1  137 --
1  138 --   {-------------------------------------------------------------------------------------------------------}
1  139 --
1  140 --
1  141 --   TDialogView = SUBCLASS OF TView  {a view which contains dialog images as well as, possibly, other things}
1  142 --
1  143 --       rootDialog:          TDialog;      {The children of this object are the constituent Dialogs of this view}
1  144 --
1  145 --       nonDialogExtent:     LRect;        {intinsic overall extent, dialog + non-dialog actually}
1  146 --
1  147 --       currentDialogImage:  TDialogImage; {which descendent owns the mouse during drag}
1  148 --
1  149 --       defaultButton:       TButton;      {which if any button is the default}
1  150 --       hitButton:           TButton;      {which Button was last chosen}
1  151 --       isShowing:           BOOLEAN;      {used to suppress meaningless screen actions for not-yet-showing box}
1  152 --
1  153 --       paintFreeBoxes:      BOOLEAN;      {whether free-checkboxes are to be painted in one sense only}
1  154 --       paintSense:          BOOLEAN;      { ... and if so, in which sense }
1  155 --       startedPainting:     BOOLEAN;      {whether we've begun to paint and hence established paintSense}
1  156 --
1  157 --       styleSheet:          TStyleSheet;  {for use by text images seen in the view}
1  158 --
1  159 --       mouseIsDown:         BOOLEAN;
1  160 --       magnetCursor:        TCursorNumber; {to force CursorAt to return this value until mouseIsDown is FALSE}
1  161 --
1  162 --   { *** Public Interface *** }
1  163 --     {Creation/Destruction}
1  164 --       FUNCTION  TDialogView.CREATE(object: TObject; heap: THeap;  itsExtentLRect: LRect; itsPanel: TPanel;
1  165 --             itsPrintManager: TPrintManager; itsRes: Point): TDialogView;
1  166 --       PROCEDURE TDialogView.Free; OVERRIDE;
1  167 --
1  168 --     {Installing, Removing, Activating, Deactivating dialogs}
1  169 --       PROCEDURE TDialogView.AddDialog(dialog: TDialog);
1  170 --       FUNCTION  TDialogView.AddNewDialog(itsKey: S4): TDialog;
1  171 --       PROCEDURE TDialogView.ActivateDialog(dialog: TDialog; whichWay: BOOLEAN);
1  172 --       PROCEDURE TDialogView.RemoveDialog(dialog: TDialog; andFree: BOOLEAN);
1  173 --       PROCEDURE TDialogView.ReplaceDialog(oldDialog, newDialog: TDialog);
1  174 --
1  175 --     {Methods which client should redefine to get a dialogView also to have non-dialog behaviour}
1  176 --       FUNCTION  TDialogView.XCursorAt(mouseLPt: LPoint): TCursorNumber; DEFAULT;
1  177 --       PROCEDURE TDialogView.XDraw; DEFAULT;
1  178 --       PROCEDURE TDialogView.XMousePress(mouseLPt: LPoint); DEFAULT;
1  179 --       PROCEDURE TDialogView.XMouseMove(mouseLPt: LPoint); DEFAULT;
1  180 --       PROCEDURE TDialogView.XMouseRelease; DEFAULT;
1  181 --
1  182 --     {Buttons and checkboxes}
1  183 --       PROCEDURE TDialogView.AbandonThatButton;
1  184 --       PROCEDURE TDialogView.ButtonPushed(button: TButton);   {normally, TDialog's ButtonPushed is used}
1  185 --       PROCEDURE TDialogView.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN);
1  186 --       PROCEDURE TDialogView.PushButton(button: TButton);     {client or ToolKit may call}
1  187 --       PROCEDURE TDialogView.SetDefaultButton(button: TButton);
1  188 --         {NB:  PushButton sets the dialogView's hitButton to the requested button, assures that it
1  189 --               is highlighted, and then calls the client's ButtonPushed method of the TDialog which
1  190 --               is the parent of the button}
1  191 --
1  192 --   { *** Private Interface ***  (Methods not expected to be redefined or called by client)}
1  193 --       FUNCTION  TDialogView.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  194 --       PROCEDURE TDialogView.Draw; OVERRIDE;
1  195 --       PROCEDURE TDialogView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  196 --       PROCEDURE TDialogView.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  197 --       PROCEDURE TDialogView.MousePress(mouseLPt: LPoint); OVERRIDE;
1  198 --       PROCEDURE TDialogView.MouseRelease; OVERRIDE;
1  199 --       PROCEDURE TDialogView.RecalcExtent; OVERRIDE;
1  200 --
1  201 --   END;  {TDialogView interface}
1  202 --
1  203 --   {-------------------------------------------------------------------------------------------------------}
1  204 --
1  205 --
1  206 --   TDialogImage = SUBCLASS OF TImage
1  207 --
1  208 --       parent:          TDialogImage;
1  209 --       isActive:        BOOLEAN;
1  210 --       isEditable:      BOOLEAN;
1  211 --       withID:          BOOLEAN;
1  212 --
1  213 --   {Creation/destruction}
1  214 --       FUNCTION  TDialogImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
1  215 --                 itsView: TView; withChildren: BOOLEAN): TDialogImage;
1  216 --
1  217 --       PROCEDURE TDialogImage.ControlHit(control: TDialogImage; toggleDirection: BOOLEAN); DEFAULT;
1  218 --       FUNCTION  TDialogImage.DownAt(mouseLPt: LPoint):  TDialogImage; DEFAULT;
1  219 --       PROCEDURE TDialogImage.Draw; OVERRIDE;
1  220 --       PROCEDURE TDialogImage.DrawJustMe; {called by Draw after children, if any, are told to draw} DEFAULT;
```

```
1  221 --       FUNCTION  TDialogImage.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  222 --       PROCEDURE TDialogImage.PrepareToAppear;
1  223 --       PROCEDURE TDialogImage.RecalcExtent;  OVERRIDE;
1  224 --       FUNCTION  TDialogImage.StillMyMouse(mouseLPt: LPoint): BOOLEAN; DEFAULT;
1  225 --
1  226 --      {The following methods are stubs, redefined in TImageWithID}
1  227 --       PROCEDURE TDialogImage.AddImage(dialogImage: TDialogImage); DEFAULT;
1  228 --       PROCEDURE TDialogImage.ActivateImage(dialogImage: TDialogImage; whichWay: BOOLEAN); DEFAULT;
1  229 --       PROCEDURE TDialogImage.BringToFront(dialogImage: TDialogImage); DEFAULT;
1  230 --       PROCEDURE TDialogImage.ComeForward; DEFAULT;
1  231 --       PROCEDURE TDialogImage.DeleteImage(dialogImage: TDialogImage; andFree: BOOLEAN); DEFAULT;
1  232 --       PROCEDURE TDialogImage.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  233 --       FUNCTION  TDialogImage.HasId(id: S255): BOOLEAN; DEFAULT;
1  234 --       FUNCTION  TDialogImage.ObjectWithIDNumber(idNumber: INTEGER): TDialogImage; DEFAULT;
1  235 --       FUNCTION  TDialogImage.ObjWithId(id: S255): TDialogImage; DEFAULT;
1  236 --       PROCEDURE TDialogImage.ReplaceImage(replacee, newValue: TDialogImage); DEFAULT;
1  237 --
1  238 --       END;
1  239 --
1  240 --
1  241 --    TImageWithID = SUBCLASS OF TDialogImage   {same interface as TDialogImage, basically}
1  242 --
1  243 --       children:   TList;  {of TDialogImage}
1  244 --       id:         TId;
1  245 --       idNumber:   INTEGER;
1  246 --
1  247 --       FUNCTION  TImageWithID.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
1  248 --                   itsView: TView; withChildren: BOOLEAN): TImageWithID;
1  249 --       FUNCTION  TImageWithID.Clone(heap: THeap): TObject;   OVERRIDE;
1  250 --       PROCEDURE TImageWithID.Free; OVERRIDE;
1  251 --
1  252 --       PROCEDURE TImageWithID.AddImage(dialogImage: TDialogImage); OVERRIDE;
1  253 --       PROCEDURE TImageWithID.ActivateImage(dialogImage: TDialogImage; whichWay: BOOLEAN); OVERRIDE;
1  254 --       PROCEDURE TImageWithID.BringToFront(dialogImage: TDialogImage); OVERRIDE;
1  255 --       FUNCTION  TImageWithID.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  256 --       PROCEDURE TImageWithID.DeleteImage(dialogImage: TDialogImage; andFree: BOOLEAN); OVERRIDE;
1  257 --       PROCEDURE TImageWithID.Draw; OVERRIDE;
1  258 --       PROCEDURE TImageWithID.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  259 --       PROCEDURE TImageWithID.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  260 --       FUNCTION  TImageWithID.HasId(id: S255): BOOLEAN; OVERRIDE;
1  261 --       PROCEDURE TImageWithID.HaveView(view: TView); OVERRIDE;
1  262 --       FUNCTION  TImageWithID.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  263 --       FUNCTION  TImageWithID.ObjectWithIDNumber(idNumber: INTEGER): TDialogImage; OVERRIDE;
1  264 --       FUNCTION  TImageWithID.ObjWithId(id: S255): TDialogImage; OVERRIDE;
1  265 --       PROCEDURE TImageWithID.OffSetBy(deltaLPt: LPoint); OVERRIDE;
1  266 --       PROCEDURE TImageWithID.RecalcExtent;   OVERRIDE;
1  267 --       PROCEDURE TImageWithID.ReplaceImage(replacee, newValue: TDialogImage); OVERRIDE;
1  268 --       FUNCTION  TImageWithID.StillMyMouse(mouseLPt : LPoint): BOOLEAN; OVERRIDE;
1  269 --
1  270 --       END;
1  271 --
1  272 --    {-------------------------------------------------------------------------------------------------------}
1  273 --
1  274 --
1  275 --    TDialog = SUBCLASS OF TImageWithID
1  276 --
1  277 --       stringKey:  TStringKey;   {essentially a unique 4-character ID by which this dialog is known}
1  278 --
1  279 --    {Creation}
1  280 --
1  281 --       FUNCTION  TDialog.CREATE(object: TObject; heap: THeap; itsKey: S4; itsView: TView): TDialog;
1  282 --
1  283 --    {Creation of the basic dialog elements:}
1  284 --
1  285 --      {Elements originating from phrase file; in each case, the text for the legend associated with the
1  286 --           component, if any, as well as a LOCATION for the component, are obtained from the same entry
1  287 --           in the phrase file, with the syntax
1  288 --
1  289 --               <text>@<h-coordinate>,<v-coordinate>
1  290 --
1  291 --            EXAMPLE:  Suppose the following 2 lines are in the Phrase File:
1  292 --
1  293 --                   449
1  294 --                   Next@430,50
1  295 --
1  296 --            If we call NewButton(449, ...), then a button is created, with the text 'Next' inside it;
1  297 --            the button is given idNumber 449, and is located at (430, 50)}
1  298 --
1  299 --    {************************** PUBLIC INTERFACE -- USE THESE METHODS ********************************}
1  300 --
1  301 --       FUNCTION  TDialog.NewButton(itsPhrase: INTEGER; itsMetrics: TButtonMetrics; sameSizedButton: TButton;
1  302 --                      itsCmdNumber: TCmdNumber): TButton;
1  303 --
1  304 --       FUNCTION  TDialog.NewCluster(itsPhrase: INTEGER): TCluster;
1  305 --
1  306 --       FUNCTION  TDialog.NewFreeCheckbox(itsPhrase: INTEGER;  boxWidth: INTEGER;
1  307 --           boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckBox;
1  308 --
1  309 --       FUNCTION  TDialog.NewInputFrame(itsPhrase: INTEGER; promptTypeStyle: TTypeStyle;
1  310 --              inputOffset: Point; inputTypeStyle: TTypeStyle;
1  311 --              maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect: BOOLEAN;
1  312 --              drawHitLRect: BOOLEAN): TInputFrame;
1  313 --
1  314 --       FUNCTION  TDialog.NewLegend(itsPhrase: INTEGER; itsTypeStyle: TTypeStyle): TLegend;
1  315 --
1  316 --       FUNCTION  TDialog.NewRowOfBoxes(itsPhrase: INTEGER; numberOfBoxes: INTEGER;
1  317 --           startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER): TCluster;
1  318 --
1  319 --      {controls}
1  320 --       PROCEDURE TDialog.ButtonPushed(button: TButton); DEFAULT; {client overrides often}
1  321 --       PROCEDURE TDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN); DEFAULT;
1  322 --                                    {client overrides sometimes}
1  323 --       PROCEDURE TDialog.ControlHit(control: TDialogImage; toggleDirection: BOOLEAN); OVERRIDE;
1  324 --       PROCEDURE TDialog.PushButton(button: TButton);      {client or ToolKit may call}
1  325 --       PROCEDURE TDialog.SelectInputFrame(inputFrame: TInputFrame);
1  326 --       PROCEDURE TDialog.SetDefaultButton(button: TButton);
1  327 --
1  328 --
1  329 --    {************************************ PRIVATE INTERFACE ************************************************}
1  330 --
```

```
1  331 --  {These methods of TDialog are largely either for internal use of the building block, or maintained for
1  332 --   backward compatability with earlier versions of the dialog building block}
1  333 --
1  334 --    {"Standard" elements:}
1  335 --      FUNCTION  TDialog.AddStdButton(itsId: S255; itsXLoc, itsYLoc: LONGINT; sameSizedButton: TButton;
1  336 --                                     itsCmdNumber: TCmdNumber): TButton;
1  337 --      PROCEDURE TDialog.AddOKButton(cmdNumber: TCmdNumber);       {OK Button}
1  338 --      PROCEDURE TDialog.AddCancelButton(cmdNumber: TCmdNumber);   {Cancel Button}
1  339 --      FUNCTION  TDialog.AddStdCluster(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCluster;
1  340 --      FUNCTION  TDialog.AddStdFreeCheckbox(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCheckBox;
1  341 --      FUNCTION  TDialog.AddStdInputFrame(itsId: S255; itsXLoc: LONGINT;
1  342 --                                         itsYLoc: LONGINT; maxInputChars : INTEGER): TInputFrame;
1  343 --      FUNCTION  TDialog.AddStdLegend(itsId: S255; itsXLoc, itsYLoc: LONGINT;
1  344 --                                     itsTypeStyle: TTypeStyle): TLegend;
1  345 --      FUNCTION  TDialog.AddSysLegend(itsId: S255; itsXLoc, itsYLoc: LONGINT): TLegend;
1  346 --
1  347 --    {General creation of dialogImages}
1  348 --      FUNCTION  TDialog.AddButton(itsId: S255; itsLocation: LPoint; itsMetrics: TButtonMetrics;
1  349 --                                  sameSizedButton: TButton; itsCmdNumber: TCmdNumber): TButton;
1  350 --
1  351 --      FUNCTION  TDialog.AddFreeCheckbox(itsID: S255; itsXLoc, itsYLoc: LONGINT; boxWidth: INTEGER;
1  352 --          boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox;
1  353 --
1  354 --      FUNCTION  TDialog.AddBigFreeCheckbox(itsId: S255; itsXLoc, itsYLoc: LONGINT): TCheckbox;
1  355 --
1  356 --      FUNCTION  TDialog.AddRowOfBoxes(itsID: S255; itsXLoc, itsYLoc: LONGINT; numberOfBoxes: INTEGER;
1  357 --          startingIDNumber: INTEGER; boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER): TCluster;
1  358 --
1  359 --      FUNCTION  TDialog.AddInputFrame(itsId: S255;
1  360 --              promptLocation: LPoint; promptTypeStyle: TTypeStyle;
1  361 --              inputLocation:  LPoint; inputTypeStyle: TTypeStyle;
1  362 --              maxInputChars: INTEGER; itsBorders: Rect; drawInputLRect: BOOLEAN;
1  363 --              drawHitLRect: BOOLEAN): TInputFrame;
1  364 --
1  365 --      FUNCTION  TDialog.DownAt(mouseLPt: LPoint): TDialogImage; OVERRIDE;
1  366 --      PROCEDURE TDialog.RecalcExtent; OVERRIDE;
1  367 --
1  368 --      END;
1  369 --
1  370 --
1  371 --  {-------------------------------------------------------------------------------------------------------}
1  372 --
1  373 --
1  374 --  TButton = SUBCLASS OF TImageWithID
1  375 --
1  376 --      cmdNumber:                TCmdNumber;
1  377 --      minWidth:                 INTEGER;
1  378 --      isHighlighted:            BOOLEAN;
1  379 --      nextSameSizedButton:      TButton;
1  380 --      legend:                   TLegend;
1  381 --      buttonMetrics:            TButtonMetrics;
1  382 --
1  383 --  {Creation/Destruction}
1  384 --      FUNCTION  TButton.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
1  385 --            itsLocation: LPoint; itsMetrics: TButtonMetrics;  sameSizedButton: TButton;
1  386 --            itsCmdNumber: TCmdNumber):  TButton;
1  387 --
1  388 --      PROCEDURE TButton.DrawJustMe; OVERRIDE;
1  389 --      PROCEDURE TButton.Highlight(highTransit: THighTransit);
1  390 --      FUNCTION  TButton.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  391 --      PROCEDURE TButton.MousePress(mouseLPt: LPoint); OVERRIDE;
1  392 --      PROCEDURE TButton.MouseRelease; OVERRIDE;
1  393 --      PROCEDURE TButton.RecalcExtent; OVERRIDE;
1  394 --      PROCEDURE TButton.Recompute(minWidth: INTEGER);
1  395 --      FUNCTION  TButton.StillMyMouse(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
1  396 --
1  397 --  END; {TButton interface}
1  398 --
1  399 --  {-------------------------------------------------------------------------------------------------------}
1  400 --
1  401 --  TCheckbox = SUBCLASS of TImageWithID
1  402 --
1  403 --      isSelected:    BOOLEAN;
1  404 --
1  405 --      rectImage:     TRectImage;      {also a child}
1  406 --      legend:        TLegend;         {if nonnil, also a child}
1  407 --
1  408 --      FUNCTION TCheckbox.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
1  409 --          itsLocation: LPoint; boxWidth: INTEGER; boxHeight: INTEGER; wantLabel: BOOLEAN;
1  410 --          labelOffset: Point; itsTypeStyle: TTypeStyle): TCheckbox;
1  411 --
1  412 --      PROCEDURE TCheckbox.ChangeLabel(newS255: S255);
1  413 --      FUNCTION  TCheckbox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  414 --      PROCEDURE TCheckbox.Draw; OVERRIDE;
1  415 --      FUNCTION  TCheckbox.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  416 --      PROCEDURE TCheckbox.MousePress(mouseLPt: LPoint); OVERRIDE;
1  417 --      PROCEDURE TCheckbox.Toggle;
1  418 --
1  419 --    END;  {TCheckbox interface}
1  420 --
1  421 --  {-------------------------------------------------------------------------------------------------------}
1  422 --
1  423 --  TCluster = SUBCLASS of TImageWithID
1  424 --
1  425 --     {children:    TList;  (of TCheckbox) }
1  426 --
1  427 --      location:    LPoint;              {only used for adding the first aligned checkbox}
1  428 --      hitBox:      TCheckbox;           {which one was just successfully queried by Hit}
1  429 --      hiLitBox:    TCheckbox;           {which one is highlighted}
1  430 --      lastBox:     TCheckBox;           {the checkbox most recently added checkbox}
1  431 --
1  432 --      FUNCTION  TCluster.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
1  433 --                                itsLocation : LPoint): TCluster;
1  434 --
1  435 --  {****** PUBLIC INTERFACE:
1  436 --   ******
1  437 --   ****** Create a cluster using TDialog.NewCluster; add checkboxes to it by calling any of the following
1  438 --   ****** three methods.  To change which box is selected in the cluster programmatically, call SelectBox
1  439 --   ******
1  440 --   ****** To find out which box is selected in a cluster, look at cluster.hiLitBox.idNumber}
```

```
1  441 --
1  442 --        FUNCTION  TCluster.NewAlignedCheckbox(itsPhrase: INTEGER; selectThisOne: BOOLEAN): TCheckbox;
1  443 --        FUNCTION  TCluster.NewCheckbox(itsPhrase: INTEGER; boxWidth: INTEGER;
1  444 --                        boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle;
1  445 --                        selectThisOne: BOOLEAN): TCheckbox;
1  446 --        PROCEDURE TCluster.AddRowOfBoxes(numberOfBoxes: INTEGER; startingIDNumber: INTEGER;
1  447 --                                boxWidth: INTEGER; boxHeight: INTEGER; boxSpacing: INTEGER);
1  448 --
1  449 --      PROCEDURE TCluster.SelectBox(checkbox: TCheckbox); {select this box, deselecting others}
1  450 --
1  451 --    {****** PRIVATE INTERFACE:
1  452 --     ******
1  453 --     ******  These remaining methods of TCluster are for primarily for internal use:}
1  454 --
1  455 --      FUNCTION  TCluster.AddAlignedCheckbox(itsId: S255; selectThisOne: BOOLEAN): TCheckbox;
1  456 --      FUNCTION  TCluster.AddCheckbox(itsID: S255; itsLocation: LPoint; boxWidth: INTEGER;
1  457 --                        boxHeight: INTEGER; wantLabel: BOOLEAN; labelOffset: Point; itsTypeStyle: TTypeStyle;
1  458 --                        selectThisOne: BOOLEAN): TCheckbox;
1  459 --      FUNCTION  TCluster.Hit(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
1  460 --      PROCEDURE TCluster.MousePress(mouseLPt: LPoint); OVERRIDE;
1  461 --      FUNCTION  TCluster.StillMyMouse(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
1  462 --
1  463 --    END; {TCluster interface}
1  464 --
1  465 --    {-------------------------------------------------------------------------------------------------------}
1  466 --
1  467 --
1  468 --    TInputFrame = SUBCLASS OF TImageWithID
1  469 --
1  470 --        textDialogImage:    TTextDialogImage;
1  471 --        prompt:             TLegend;
1  472 --
1  473 --        borders:            Rect;
1  474 --
1  475 --        drawInputLRect:     BOOLEAN;  {whether or not to draw a faint box around the input LRect}
1  476 --        drawHitLRect:       BOOLEAN;  {whether or not to frame the hit rectangle}
1  477 --        maxInputChars:      INTEGER;
1  478 --        inputTypeStyle:     TTypeStyle;
1  479 --
1  480 --        FUNCTION  TInputFrame.CREATE(object: TObject; heap: THeap; itsId: S255; itsView: TView;
1  481 --              promptLocation: LPoint; promptTypeStyle: TTypeStyle;
1  482 --              inputLocation:  LPoint; inputTypeStyle: TTypeStyle;  maxInputChars: INTEGER;
1  483 --              itsBorders: Rect; drawInputLRect: BOOLEAN; drawHitLRect: BOOLEAN
1  484 --              ): TInputFrame;
1  485 --
1  486 --
1  487 --    { ********************************************** PUBLIC INTERFACE ********************************** }
1  488 --
1  489 --       {Create an input frame by calling TDialog.NewInputFrame; use GetContents and SupplantContents
1  490 --        to find out what has been typed, and to change what appears in the typing area}
1  491 --
1  492 --        PROCEDURE TInputFrame.GetContents(VAR theStr: S255);      {inspect current frame contents}
1  493 --        PROCEDURE TInputFrame.SupplantContents(newStr: S255);     {change current frame contents}
1  494 --
1  495 --    { ********************************************** PRIVATE INTERFACE ********************************* }
1  496 --
1  497 --        FUNCTION  TInputFrame.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  498 --        PROCEDURE TInputFrame.Draw; OVERRIDE;
1  499 --        FUNCTION  TInputFrame.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  500 --        PROCEDURE TInputFrame.MousePress(mouseLPt: LPoint); OVERRIDE;
1  501 --        PROCEDURE TInputFrame.RecalcExtent; OVERRIDE;
1  502 --        FUNCTION  TInputFrame.StillMyMouse(mouseLPt: LPoint): BOOLEAN; OVERRIDE;
1  503 --
1  504 --     END;    {TInputFrame interface}
1  505 --
1  506 --    {-------------------------------------------------------------------------------------------------------}
1  507 --
1  508 --    TLegend = SUBCLASS OF TDialogImage
1  509 --
1  510 --            location:           LPoint;
1  511 --            paragraph:          TParagraph;
1  512 --            wouldBeDraggable:   BOOLEAN;        {whether, during layout, it should itself be draggable}
1  513 --            usesSysFont:        BOOLEAN;        {whether it is in system font -- a special case}
1  514 --
1  515 --        FUNCTION  TLegend.CREATE(object: TObject; heap: THeap; itsChars: S255; itsView: TView;
1  516 --            itsLocation: LPoint; itsTypeStyle: TTypeStyle): TLegend;
1  517 --        PROCEDURE TLegend.Free; OVERRIDE;
1  518 --
1  519 --    { ********************************************** PUBLIC INTERFACE ********************************** }
1  520 --
1  521 --        PROCEDURE TLegend.ChangeToPhrase(newPhrase: INTEGER); {for getting new text from phrase file}
1  522 --        PROCEDURE TLegend.ChangeString(newString: S255);      {for getting new text from a string}
1  523 --        PROCEDURE TLegend.GetString(VAR itsString: S255); {determine current chars residing in the legend}
1  524 --
1  525 --    { ********************************************** PRIVATE INTERFACE ********************************* }
1  526 --        PROCEDURE TLegend.Draw; OVERRIDE;
1  527 --        PROCEDURE TLegend.GetBoxRight;    {sets extent based on current chars & location}
1  528 --        FUNCTION  TLegend.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  529 --        PROCEDURE TLegend.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  530 --        PROCEDURE TLegend.RecalcExtent; OVERRIDE;
1  531 --
1  532 --        END;
1  533 --
1  534 --    { ------------------------------ classes implemented in file UDialog3 ------------------------------}
1  535 --
1  536 --
1  537 --
1  538 --    TPicObject = SUBCLASS OF TImageWithID {An Object which holds a QD Picture File} {CAUTION: totally untested}
1  539 --
1  540 --        picture:          PicHandle;
1  541 --        boxAtCreation:    Rect;    {need to get itsView parameter into all these guys}
1  542 --
1  543 --        FUNCTION  TPicObject.CREATE(object: TObject; heap: THeap; itsId: S255;
1  544 --             itsView: TView; itsLocation: LPoint; itsPicHandle: PicHandle): TPicObject;
1  545 --        PROCEDURE TPicObject.Free; OVERRIDE;
1  546 --
1  547 --        PROCEDURE TPicObject.Draw; OVERRIDE;
1  548 --
1  549 --        END;
1  550 --
```

```
1  551 --  {-------------------------------------------------------------------------------------------------------------}
1  552 --
1  553 --    TRectImage = SUBCLASS OF TDialogImage   {a rectangle packaged as a object}
1  554 --
1  555 --         penState: PenState;
1  556 --
1  557 --       FUNCTION TRectImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
1  558 --           itsView: TView; itsPenState: PenState; withChildren: BOOLEAN): TRectImage;
1  559 --
1  560 --       PROCEDURE TRectImage.Draw; OVERRIDE;
1  561 --       FUNCTION  TRectImage.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  562 --       END;
1  563 --
1  564 --  {-------------------------------------------------------------------------------------------------------------}
1  565 --
1  566 --
1  567 --  TTextDialogImage = SUBCLASS OF TImageWithID
1  568 --
1  569 --        textImage:          TTextImage;
1  570 --        wouldBeDraggable:   BOOLEAN;
1  571 --        refCount:           INTEGER;
1  572 --
1  573 --       FUNCTION  TTextDialogImage.CREATE(object: TObject; heap: THeap; itsExtent: LRect; itsId: S255;
1  574 --                     itsView: TView; itsTypeStyle: TTypeStyle;
1  575 --                     itsInitialChars: S255): TTextDialogImage;
1  576 --       PROCEDURE TTextDialogImage.Free; OVERRIDE;
1  577 --
1  578 --       PROCEDURE TTextDialogImage.ChangeRefCountBy(delta: INTEGER);
1  579 --       FUNCTION  TTextDialogImage.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  580 --       PROCEDURE TTextDialogImage.Draw; OVERRIDE;
1  581 --       FUNCTION  TTextDialogImage.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  582 --       PROCEDURE TTextDialogImage.MousePress(mouseLPt: LPoint); OVERRIDE;
1  583 --       PROCEDURE TTextDialogImage.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  584 --        END;
1  585 --
1  586 --  {-------------------------------------------------------------------------------------------------------------}
1  587 --
1  588 --  TFrameSelection = SUBCLASS OF TSelection  {the phony selection covering TextSelection in an input frame}
1  589 --
1  590 --       inputFrame: TInputFrame;   {the input frame in which the selection occurs}
1  591 --
1  592 --       FUNCTION  TFrameSelection.CREATE(object: TObject; heap: THeap; itsInputFrame: TInputFrame)
1  593 --                                    : TFrameSelection;
1  594 --
1  595 --       FUNCTION  TFrameSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; OVERRIDE;
1  596 --       PROCEDURE TFrameSelection.KeyChar(ch: CHAR); OVERRIDE;
1  597 --       PROCEDURE TFrameSelection.KeyEnter(dh, dv: INTEGER); OVERRIDE;
1  598 --       PROCEDURE TFrameSelection.KeyReturn; OVERRIDE;
1  599 --       PROCEDURE TFrameSelection.KeyTab(fBackward: BOOLEAN);  OVERRIDE;
1  600 --       PROCEDURE TFrameSelection.MousePress(mouseLPt: LPoint);  OVERRIDE;
1  601 --       PROCEDURE TFrameSelection.PerformCommand(command: TCommand; cmdPhase: TCmdPhase); OVERRIDE;
1  602 --       PROCEDURE TFrameSelection.Restore; OVERRIDE;
1  603 --
1  604 --  END;  {TFrameSelection interface}
1  605 --
1  606 --
1  607 --  TPlannerView = SUBCLASS OF TDialogView  {a view within which images are laid out}
1  608 --
1  609 --    {Variables}
1  610 --
1  611 --       viewBeingPlanned:    TView;
1  612 --
1  613 --       allowSketching:      BOOLEAN;    {for internal use of the layout mechanism}
1  614 --       retainPickedBox:     BOOLEAN;
1  615 --       currentLayoutBox:    TLayoutBox;
1  616 --
1  617 --    {Creation/Destruction}
1  618 --       FUNCTION  TPlannerView.CREATE(object: TObject; heap: THeap; itsViewBeingPlanned: TView;
1  619 --                 itsPanel: TPanel; itsAllowSketching: BOOLEAN; itsRetainPickedBox: BOOLEAN): TPlannerView;
1  620 --       PROCEDURE TPlannerView.Init(itsListOfImages: TList);
1  621 --       FUNCTION  TPlannerView.NewLayoutBox(image: TImage): TLayoutBox; {return NIL if element not to be shown}
1  622 --
1  623 --       PROCEDURE TPlannerView.Free; OVERRIDE;
1  624 --
1  625 --    {Display}
1  626 --       PROCEDURE TPlannerView.Draw; OVERRIDE;
1  627 --
1  628 --    {Mouse Tracking}
1  629 --       FUNCTION  TPlannerView.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  630 --       PROCEDURE TPlannerView.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  631 --       PROCEDURE TPlannerView.MousePress(mouseLPt: LPoint); OVERRIDE;
1  632 --       PROCEDURE TPlannerView.MouseRelease; OVERRIDE;
1  633 --
1  634 --    {Enumeration of components}
1  635 --       PROCEDURE TPlannerView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  636 --
1  637 --       END;
1  638 --
1  639 --  {-------------------------------------------------------------------------------------------------------------}
1  640 --
1  641 --
1  642 --       TLayoutBox = SUBCLASS OF TImageWithID
1  643 --
1  644 --           {Variables}
1  645 --           manipulee:                  TImage;
1  646 --           titleTab:                   TTitleTab;
1  647 --
1  648 --           suppressDrawingManipulee:   BOOLEAN;
1  649 --
1  650 --           isResizable:                BOOLEAN;
1  651 --           borders:                    Rect;
1  652 --           wouldMakeSelection:         BOOLEAN;    {client must directly set if not wanting default 'FALSE'}
1  653 --
1  654 --           isDraggable:                BOOLEAN;
1  655 --           shouldFrame:                BOOLEAN;
1  656 --
1  657 --           hasDraggee:                 BOOLEAN;
1  658 --
1  659 --         {Creation/Destruction}
1  660 --           FUNCTION  TLayoutBox.CREATE(object: TObject; heap: THeap; baseExtent: LRect; itsID: S255;
```

```
1  661 --              itsParent: TLayoutBox; itsView: TView; itsManipulee: TImage; itsBorders: Rect;
1  662 --              itsResizable: BOOLEAN; itsSuppression: BOOLEAN; withChildren: BOOLEAN): TLayoutBox;
1  663 --          PROCEDURE TLayoutBox.Free; OVERRIDE;
1  664 --
1  665 --        {Change and Display}
1  666 --          PROCEDURE TLayoutBox.ChangeDragState(enteringDrag: BOOLEAN);
1  667 --          PROCEDURE TLayoutBox.ConsiderMouse(mouseLPt: LPoint; VAR madeSelection: BOOLEAN;
1  668 --                                             VAR pickedLayoutBox: TLayoutBox); DEFAULT;
1  669 --          FUNCTION  TLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  670 --          PROCEDURE TLayoutBox.Draw; OVERRIDE;
1  671 --          PROCEDURE TLayoutBox.DrawJustMe; OVERRIDE;
1  672 --          PROCEDURE TLayoutBox.FreeManipulee;
1  673 --          PROCEDURE TLayoutBox.Highlight(highTransit: THighTransit);
1  674 --          PROCEDURE TLayoutBox.MousePress(mouseLPT: LPoint); OVERRIDE;
1  675 --          PROCEDURE TLayoutBox.Move(deltaLPt: LPoint); DEFAULT;
1  676 --          FUNCTION  TLayoutBox.NoTitleTab(heap: THeap): TTitleTab;
1  677 --          PROCEDURE TLayoutBox.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  678 --          PROCEDURE TLayoutBox.OffsetLayoutBoxBy(deltaLPt: LPoint; textImageAsWell: BOOLEAN); DEFAULT;
1  679 --          PROCEDURE TLayoutBox.RecalcExtent; OVERRIDE;
1  680 --          PROCEDURE TLayoutBox.Resize(newExtent: LRect); OVERRIDE;
1  681 --          PROCEDURE TLayoutBox.TabGrabbed; DEFAULT;
1  682 --
1  683 --          END;
1  684 --
1  685 --      TLegendLayoutBox = SUBCLASS OF TLayoutBox  {manipulee is a TLegend}
1  686 --
1  687 --          textDialogImage:  TTextDialogImage;
1  688 --
1  689 --        {Creation/Destruction}
1  690 --          FUNCTION  TLegendLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView; itsLegend: TLegend
1  691 --                  ): TLegendLayoutBox;
1  692 --
1  693 --          FUNCTION  TLegendLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  694 --          PROCEDURE TLegendLayoutBox.Draw; OVERRIDE;
1  695 --          PROCEDURE TLegendLayoutBox.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  696 --          PROCEDURE TLegendLayoutBox.OffsetLayoutBoxBy(deltaLPt: LPoint; textImageAsWell: BOOLEAN); OVERRIDE;
1  697 --            {use of the second argument is strange and non self-explanatory; comments in the internal
1  698 --             documentation may help.  Nobody should be calling this old boy from outside, anyway}
1  699 --          PROCEDURE TLegendLayoutBox.MousePress(mouseLPT: LPoint); OVERRIDE;
1  700 --          PROCEDURE TLegendLayoutBox.RecalcExtent; OVERRIDE;
1  701 --
1  702 --          END;
1  703 --
1  704 --
1  705 --      TButtonLayoutBox = SUBCLASS OF TLayoutBox  {manipulee is a TButton}
1  706 --
1  707 --        {Variables}
1  708 --          nextSameSizedBox:    TButtonLayoutBox;
1  709 --          oldLegendTopLeft:    LPoint;
1  710 --
1  711 --        {Creation/Destruction}
1  712 --          FUNCTION  TButtonLayoutBox.CREATE(object: TObject; heap: THeap; itsButton: TButton;
1  713 --                                            itsView: TView): TButtonLayoutBox;
1  714 --
1  715 --        {Other Methods}
1  716 --          PROCEDURE TButtonLayoutBox.ConsiderMouse(mouseLPt: LPoint; VAR madeSelection: BOOLEAN;
1  717 --                                                   VAR pickedLayoutBox: TLayoutBox); OVERRIDE;
1  718 --          FUNCTION  TButtonLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  719 --          PROCEDURE TButtonLayoutBox.DrawJustMe; OVERRIDE;
1  720 --          PROCEDURE TButtonLayoutBox.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  721 --          PROCEDURE TButtonLayoutBox.RecalcExtent; OVERRIDE;
1  722 --          PROCEDURE TButtonLayoutBox.RecalcJustMe;
1  723 --
1  724 --          END;
1  725 --
1  726 --
1  727 --      TTitleTab = SUBCLASS OF TImage
1  728 --
1  729 --          layoutBox:           TLayoutBox;
1  730 --          legend:              TLegend;
1  731 --          shouldDrawLegend:    BOOLEAN;  {FALSE if string is too wide to fit}
1  732 --
1  733 --          FUNCTION  TTitleTab.CREATE(object: TObject; heap: THeap; itsLayoutBox: TLayoutBox; itsHeight: INTEGER;
1  734 --              itsCaption: S255): TTitleTab;
1  735 --          PROCEDURE TTitleTab.Free; OVERRIDE;
1  736 --
1  737 --          PROCEDURE TTitleTab.Draw; OVERRIDE;
1  738 --          PROCEDURE TTitleTab.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  739 --          PROCEDURE TTitleTab.Resize(newExtent: LRect); OVERRIDE;
1  740 --          END;
1  741 --
1  742 --
1  743 --      TLayPickSelection = SUBCLASS OF TSelection
1  744 --
1  745 --        {Variables}
1  746 --          layoutBox:   TLayoutBox;
1  747 --
1  748 --          FUNCTION  TLayPickSelection.CREATE(object: TObject; heap: THeap; itsView: TPlannerView;
1  749 --                        itsKind: INTEGER; itsLayoutBox: TLayoutBox; itsAnchorLPt: LPoint): TLayPickSelection;
1  750 --
1  751 --          FUNCTION  TLayPickSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
1  752 --                                                 : BOOLEAN; OVERRIDE;
1  753 --          PROCEDURE TLayPickSelection.Deselect; OVERRIDE;
1  754 --          PROCEDURE TLayPickSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1  755 --          PROCEDURE TLayPickSelection.KeyClear; OVERRIDE;
1  756 --          PROCEDURE TLayPickSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  757 --          PROCEDURE TLayPickSelection.MouseRelease; OVERRIDE;
1  758 --          PROCEDURE TLayPickSelection.Restore; OVERRIDE;
1  759 --
1  760 --          END;
1  761 --
1  762 --
1  763 --      TLayMoveCmd = SUBCLASS OF TCommand
1  764 --
1  765 --        {Variables}
1  766 --          layoutBox:   TLayoutBox;
1  767 --
1  768 --          hOffset:     LONGINT;
1  769 --          vOffset:     LONGINT;
1  770 --
```

```
1  771 --          {Creation}
1  772 --            FUNCTION  TLayMoveCmd.CREATE(object: TObject; heap: THeap; itsLayoutBox: TLayoutBox;
1  773 --                                         itsHOffset, itsVOffset: LONGINT): TLayMoveCmd;
1  774 --
1  775 --          {Command Execution}
1  776 --            PROCEDURE  TLayMoveCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  777 --
1  778 --            END;
1  779 --
1  780 --        TEditLegendSelection = SUBCLASS OF TSelection
1  781 --
1  782 --          {Variables}
1  783 --           legendLayoutBox: TLegendLayoutBox;
1  784 --           hostLegend:      TLegend;
1  785 --           textDialogImage: TTextDialogImage;
1  786 --           suppressHost:    BOOLEAN;
1  787 --           tripleClick:     BOOLEAN; {+SW+}
1  788 --
1  789 --          {Creation/Destruction}
1  790 --           FUNCTION  TEditLegendSelection.CREATE(object: TObject; heap: THeap; itsLegendLayoutBox:
1  791 --                                                 TLegendLayoutBox; itsAnchorLPt: LPoint): TEditLegendSelection;
1  792 --           FUNCTION  TEditLegendSelection.Clone(heap: THeap): TObject;  OVERRIDE;
1  793 --           PROCEDURE TEditLegendSelection.Deselect; OVERRIDE;
1  794 --           PROCEDURE TEditLegendSelection.Free; OVERRIDE;
1  795 --
1  796 --          {Udders}
1  797 --           FUNCTION  TEditLegendSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
1  798 --                                                       : BOOLEAN; OVERRIDE;
1  799 --           PROCEDURE TEditLegendSelection.KeyBack(fWord: BOOLEAN); OVERRIDE;
1  800 --           PROCEDURE TEditLegendSelection.KeyChar(ch: CHAR); OVERRIDE;
1  801 --           PROCEDURE TEditLegendSelection.KeyEnter(dh, dv: INTEGER); OVERRIDE;
1  802 --           PROCEDURE TEditLegendSelection.KeyReturn; OVERRIDE;
1  803 --           PROCEDURE TEditLegendSelection.MouseMove(mouseLPt: LPoint); OVERRIDE; {+SW+}
1  804 --           PROCEDURE TEditLegendSelection.MousePress(mouseLPt: LPoint); OVERRIDE; {+SW+}
1  805 --           PROCEDURE TEditLegendSelection.MouseRelease; OVERRIDE; {+SW+}
1  806 --           FUNCTION  TEditLegendSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  807 --           PROCEDURE TEditLegendSelection.PerformCommand(command: TCommand; cmdPhase: TCmdPhase);  OVERRIDE;
1  808 --           PROCEDURE TEditLegendSelection.Restore; OVERRIDE;
1  809 --           PROCEDURE TEditLegendSelection.Reveal(asMuchAsPossible: BOOLEAN);  OVERRIDE;
1  810 --
1  811 --           END;
1  812 --
1  813 --
1  814 --        TDialogDesignWindow = SUBCLASS OF TDialogWindow
1  815 --
1  816 --            hostWindow:          TWindow;
1  817 --            hostDialogView:      TDialogView;
1  818 --            fromDialogBox:       BOOLEAN;
1  819 --
1  820 --         FUNCTION  TDialogDesignWindow.CREATE(object: TObject; heap: THeap;
1  821 --                   itsHostDialogView: TDialogView): TDialogDesignWindow;
1  822 --
1  823 --         FUNCTION  TDialogDesignWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
1  824 --                                                    : BOOLEAN; OVERRIDE;
1  825 --         FUNCTION  TDialogDesignWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  826 --         PROCEDURE TDialogDesignWindow.RelinquishControl;
1  827 --         PROCEDURE TDialogDesignWindow.Resize(moving: BOOLEAN); OVERRIDE;
1  828 --         PROCEDURE TDialogDesignWindow.SeizeControl;
1  829 --
1  830 --         END;
1  831 --
1  832 --
1  833 --
1  834 --    { ------------------------------ classes implemented in file UDialog4 -------------------------------}
1  835 --
1  836 --
1  837 --    TStdPrintManager = SUBCLASS OF TPrintManager
1  838 --
1  839 --        FUNCTION  TStdPrintManager.CREATE(object: TObject; heap: THeap): TStdPrintManager;
1  840 --
1  841 --        PROCEDURE TStdPrintManager.EnterPageEditting; OVERRIDE;
1  842 --        PROCEDURE TStdPrintManager.Init(itsMainView: TView; itsDfltMargins: LRect); OVERRIDE;
1  843 --        PROCEDURE TStdPrintManager.ReactToPrinterChange; OVERRIDE;
1  844 --        PROCEDURE TStdPrintManager.SetDfltHeadings;  OVERRIDE;
1  845 --
1  846 --        END;
1  847 --
1  848 --    TLegendHeading = SUBCLASS OF THeading
1  849 --
1  850 --        masterLegend:    TLegend;
1  851 --        currentLegend:   TLegend;
1  852 --
1  853 --        topToBaseline:   INTEGER;  {offset from box top to baseline}
1  854 --        borders:         Rect;     {size by which box exceeds legend's extent}
1  855 --
1  856 --      {Creation/Destruction}
1  857 --        FUNCTION  TLegendHeading.CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
1  858 --            itsString: S255; itsTypeStyle: TTypeStyle;
1  859 --            itsPageAlignment: TPageAlignment; itsOffsetFromAlignment: LPoint;
1  860 --            itsBorders: Rect): TLegendHeading;
1  861 --        PROCEDURE  TLegendHeading.Free; OVERRIDE;
1  862 --
1  863 --      {Nyingine}
1  864 --        PROCEDURE TLegendHeading.AdjustForPage(pageNumber: LONGINT; editing: BOOLEAN); OVERRIDE;
1  865 --        PROCEDURE TLegendHeading.Draw; OVERRIDE;
1  866 --        FUNCTION  TLegendHeading.LaunchLayoutBox(view: TView): TImage; OVERRIDE;
1  867 --        PROCEDURE TLegendHeading.OffsetBy(deltaLPt: LPoint); OVERRIDE;
1  868 --        PROCEDURE TLegendHeading.RecalcExtent; OVERRIDE;
1  869 --        FUNCTION  TLegendHeading.ShouldFrame: BOOLEAN; OVERRIDE;
1  870 --
1  871 --        END;
1  872 --
1  873 --
1  874 --    TPageDesignWindow = SUBCLASS OF TDialogWindow
1  875 --
1  876 --        hostView:        TView;  {the view whose pages are being designed in this dialog}
1  877 --        layoutPanel:     TPanel; {my controlPanel is the status panel}
1  878 --
1  879 --        FUNCTION  TPageDesignWindow.CREATE(object: TObject; heap: THeap; itsHostView: TView): TPageDesignWindow;
1  880 --
```

```
1  881 --       PROCEDURE TPageDesignWindow.Disappear;  OVERRIDE;
1  882 --       FUNCTION  TPageDesignWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;  OVERRIDE;
1  883 --
1  884 --       END;
1  885 --
1  886 --
1  887 --    TPagePlannerView = SUBCLASS OF TPlannerView
1  888 --
1  889 --       FUNCTION TPagePlannerView.CREATE(object: TObject; heap: THeap; itsPrintManager: TPrintManager;
1  890 --                      itsPanel: TPanel): TPagePlannerView;
1  891 --
1  892 --       PROCEDURE TPagePlannerView.Draw; OVERRIDE;
1  893 --
1  894 --       END;
1  895 --
1  896 --
1  897 --    TPageStatusDialog = SUBCLASS OF TDialog
1  898 --
1  899 --       currentHeading:      THeading;
1  900 --
1  901 --       oddEvenCluster:      TCluster;
1  902 --       minPageFrame:        TInputFrame;
1  903 --       maxPageFrame:        TInputFrame;
1  904 --       alignCluster:        TCluster;
1  905 --       unitsCluster:        TCluster;
1  906 --       marginTitle:         TLegend;
1  907 --
1  908 --       leftCluster:         TCluster;
1  909 --       topCluster:          TCluster;
1  910 --       rightCluster:        TCluster;
1  911 --       bottomCluster:       TCluster;
1  912 --
1  913 --     {Creation/Destruction}
1  914 --       FUNCTION  TPageStatusDialog.CREATE(object: TObject; heap: THeap; itsPanel: TPanel): TPageStatusDialog;
1  915 --
1  916 --     {Sonst}
1  917 --       PROCEDURE TPageStatusDialog.ButtonPushed(button: TButton); OVERRIDE;
1  918 --       PROCEDURE TPageStatusDialog.CheckboxHit(checkbox: TCheckbox; toggleDirection: BOOLEAN); OVERRIDE;
1  919 --       FUNCTION  TPageStatusDialog.DownAt(mouseLPt: LPoint): TDialogImage; OVERRIDE;
1  920 --       PROCEDURE TPageStatusDialog.Draw; OVERRIDE;
1  921 --       PROCEDURE TPageStatusDialog.InspectHeadingParms(VAR oddOnly, evenOnly: BOOLEAN;
1  922 --                                       VAR pageAlignment: TPageAlignment; VAR minPage, maxPage: LONGINT);
1  923 --       PROCEDURE TPageStatusDialog.SetHeadingParms(oddOnly, evenOnly: BOOLEAN;
1  924 --                                       pageAlignment: TPageAlignment;  minPage, maxPage: LONGINT);
1  925 --       END;
1  926 --
1  927 --
1  928 --    TPageLayoutBox = SUBCLASS OF TLayoutBox
1  929 --
1  930 --
1  931 --     {Creation/Destruction}
1  932 --       FUNCTION  TPageLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView; itsHeading: THeading;
1  933 --                      itsResizable: BOOLEAN): TPageLayoutBox;
1  934 --
1  935 --       PROCEDURE TPageLayoutBox.FreeManipulee; OVERRIDE;
1  936 --       PROCEDURE TPageLayoutBox.TabGrabbed; OVERRIDE;
1  937 --       END;
1  938 --
1  939 --
1  940 --    TLgHdngLayoutBox = SUBCLASS OF TPageLayoutBox
1  941 --
1  942 --       legendLayoutBox: TLegendLayoutBox;
1  943 --
1  944 --       FUNCTION  TLgHdngLayoutBox.CREATE(object: TObject; heap: THeap; itsView: TView;
1  945 --           itsLegendHeading: TLegendHeading): TLgHdngLayoutBox;
1  946 --
1  947 --       FUNCTION  TLgHdngLayoutBox.CursorAt(mouseLPt: LPoint): TCursorNumber; OVERRIDE;
1  948 --       PROCEDURE TLgHdngLayoutBox.Draw; OVERRIDE;
1  949 --       PROCEDURE TLgHdngLayoutBox.MousePress(mouseLPT: LPoint); OVERRIDE;
1  950 --       PROCEDURE TLgHdngLayoutBox.Move(deltaLPt: LPoint); OVERRIDE;
1  951 --       PROCEDURE TLgHdngLayoutBox.RecalcExtent; OVERRIDE;
1  952 --
1  953 --       END;
1  954 --
1  955 --
1  956 --    VAR
1  957 --       stdFrameBorders:        Rect;          {extra space around an input-frame and its text}
1  958 --       stdHdngBorders:         Rect;          {extra space around a standard heading}
1  959 --       stdHdngTypeStyle:       TTypeStyle;    {tile 12 monospaced, normal faces, for titles}
1  960 --       stdIDBorders:           Rect;          {a title tab with string, and a small border on the other 3 sides}
1  961 --       stdInputTypeStyle:      TTypeStyle;    {std input font/faces}
1  962 --       stdFrmeOffset:          Point;         {std distance between input frame's prompt and input rect}
1  963 --       stdLabelOffset:         Point;         {offset from top-left corner of a checkbox to leftmost pt of
1  964 --                                               baseline of label}
1  965 --       stdPlainBorders:        Rect;          {a slim captionless title tab, and a small border on the other
1  966 --                                               3 sides}
1  967 --       stdThinBorders:         Rect;          {a slim captionless title tab above; no other borders}
1  968 --       titleTypeStyle:         TTypeStyle;    {tile 15 monospaced, for titles of layout boxes}
1  969 --       {NB: All the above are initialized in the creation block of TDialogWindow}
1  970 --
1  971 --       stdButtonMetrics:       TButtonMetrics; {reinitialized in TDialog.CREATE each time}
1  972 --
1  973 --
1  974 --    {Unit-Global Procedures}
1  975 --
1  976 --    FUNCTION  NewStdDialogWindow(heap: THeap; itsHeight: INTEGER; itsKeyResponse, itsMenuResponse,
1  977 --                  itsDownInMainWindowResponse: TDiResponse): TDialogWindow;
1  978 --                {sets up a standard, nonresizable, dialogWindow, and installs a single Panel into it, into
1  979 --                 which it installs a single DialogView}
1  980 --
1  981 --    FUNCTION  NewStdLegend(heap: THeap; itsChars: S255; itsXLoc, itsYLoc: LONGINT; itsView: TView;
1  982 --                           itsTypeStyle: TTypeStyle): TLegend;
1  983 --
1  984 --    FUNCTION  NewSysLegend(heap: THeap; itsChars: S255; itsXLoc, itsYLoc: LONGINT; itsView: TView): TLegend;
1  985 --
1  986 --    PROCEDURE SetParaExtent(paragraph: TParagraph; view: TView; location: LPoint; VAR extentLRect: LRect);
1  987 --
1  988 --    PROCEDURE LRectAddBorders(baseLRect: LRect; borders: Rect; VAR resultLRect: LRect);
1  989 --
1  990 --    PROCEDURE GetTextAndLocation(phraseNumber: INTEGER; VAR itsChars: S255; VAR itsLocation: LPoint);
```

```
1  991 --
1  992 --
1  993 --   IMPLEMENTATION
1  994 --
2    1 --
2    2 --
1  995 --   {$I LIBTK/UDialog2}    {dialogs}
3    1 --
3    2 --
1  996 --   {$I LIBTK/UDialog3}    {layout}
4    1 --
4    2 --
1  997 --   {$I LIBTK/UDialog4}    {page margins}
1  998 --
1  999 --   {**********
1 1000 --   {$I UDialog2}    {dialogs}
1 1001 --   {$I UDialog3}    {layout}
1 1002 --   {$I UDialog4}    {page margins}
1 1003 --   **********}
1 1004 --
1 1005 --   END.  {unit UDialog}
1 1006 --
```

```
  TTextDialogImage  470 ( 1)  567*( 1)  575 ( 1)  687 ( 1)  785 ( 1)
  TTextImage        569 ( 1)
  TTitleTab         646 ( 1)  676 ( 1)  727*( 1)  734 ( 1)
  TTypeStyle         92 ( 1)  478 ( 1)  959 ( 1)  961 ( 1)  968 ( 1)
  TView             141 ( 1)  611 ( 1)  876 ( 1)
  TWindow           816 ( 1)
  typeStyle          92*( 1)

-U-
  UABC               30*( 1)
  UDialog             7*( 1)
  UDialogVersion     36*( 1)
  UDraw              28*( 1)
  UFont              25*( 1)
  unitsCluster      905*( 1)
  UObject            23*( 1)
  usesSysFont       513*( 1)
  UText              32*( 1)
  UTKUniversalText   31*( 1)

-V-
  viewBeingPlanned  611 ( 1)
  vOffset           769*( 1)

-W-
  withID            211*( 1)
  wouldBeDraggable  512*( 1)  570*( 1)
  wouldMakeSelecti  652*( 1)

-X-
  XCursorAt         176*( 1)
  XDraw             177*( 1)
  XMouseMove        179*( 1)
  XMousePress       178*( 1)
  XMouseRelease     180*( 1)

*** End Xref: 305 id's   657 references      [417544 bytes/4694 id's/42805 refs]
```