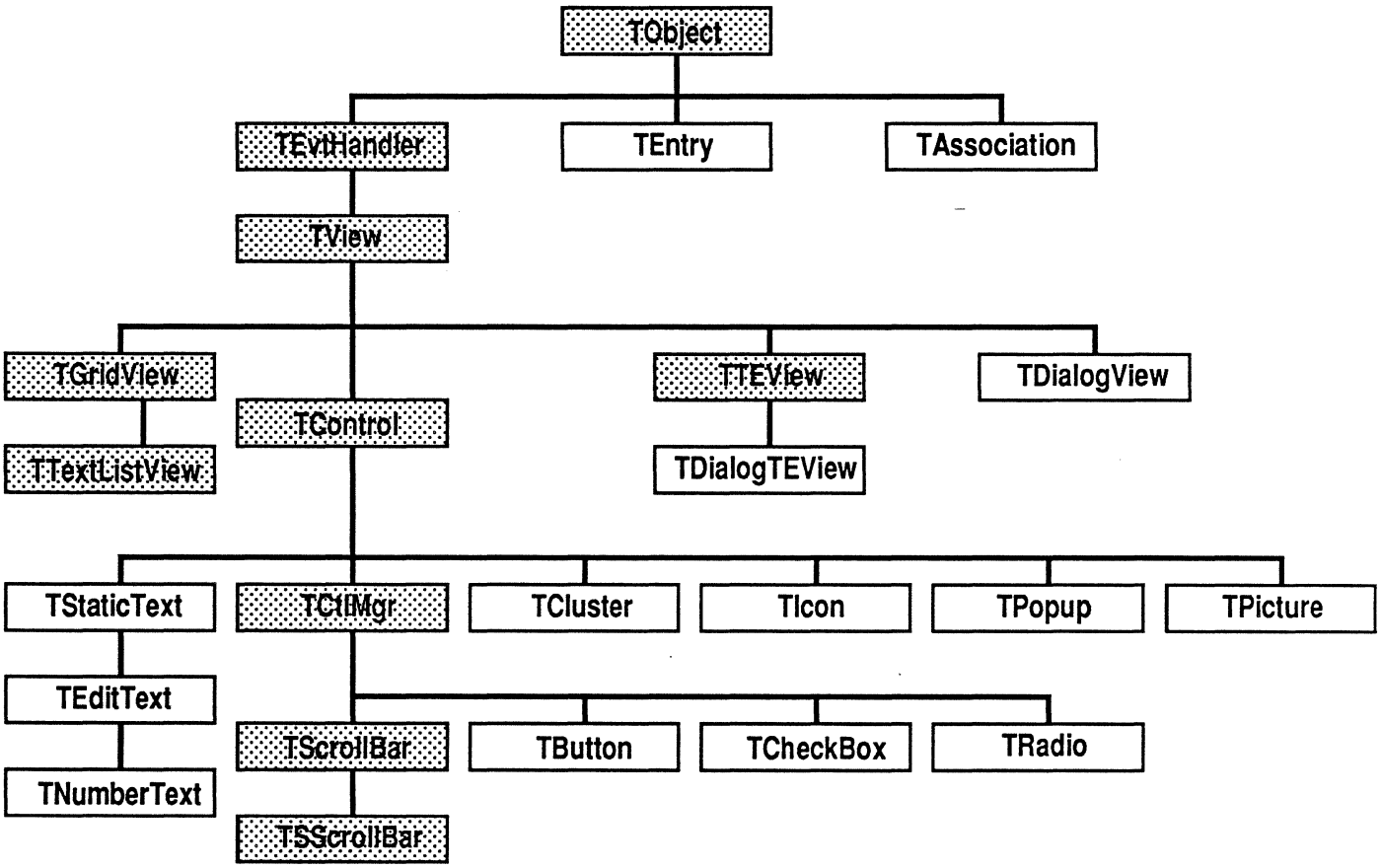
MacApp 2.0b5 UDialog Release Notes

Russ Wetmore

Overview

This document describes the classes that make up MacApp's "dialog" implementation. These classes primarily consist of a set of view classes that implement the kinds of views one often sees in Macintosh dialogs. (These views can be used in any MacApp window, regardless of whether you consider it a dialog.) The new classes can be created from resource templates.

This is the hierarchy of the classes involved. (Classes not described in this document are shown in gray.)



Global Constants

Choices

```
mOKHit                  = 1;
mCancelHit              = 2;
mButtonHit              = 3;
mCheckBoxHit            = 4;
mClusterHit             = 5;
mEditTextHit            = 6;
mIconHit                = 7;
mListItemHit            = 8;
mListScrollBarHit       = 9;
mPictureHit             = 10;
mPopupHit               = 11;
mRadioHit               = 12;
mStaticTextHit          = 13;
mHScrollBarHit          = 14;
mVScrollBarHit          = 15;
mEditTabKey             = 16;
mEditReturnKey          = 17;
mEditEnterKey           = 18;
mPatternHit             = 19;
mControlHit             = 20;
```

View Template Identifiers

```
kNoIdentifier           = '    ';
kNoTemplate             = -1;
kNoResource             = -1;
```

Miscellaneous

```
kYesButton              = 1;
kNoButton               = 3;
kPreferColor            = TRUE;
kControlOn              = TRUE;
kRedraw                 = TRUE;
kDontRedraw             = FALSE;
kFrame                  = [adnLineTop, adnLineLeft,
                           adnLineBottom, adnLineRight];
Chr00                   = CHR(0);
Chr1F                   = CHR($1F);
kMaxTEWidth             = $2B0;
```

Global Types

```
IDType          = ResType;
```

```
adornPieces  = ( adnLineTop,          Draw a line at the top of the extent.
                 adnLineLeft,         Draw a line on the left side of the extent.
                 adnLineBottom,       Draw a line at the bottom of the extent.
                 adnLineRight,        Draw a line on the right side of the extent.
                 adnDummy,            Placeholder (replaces adnPatFill).
                 adnOval,             Do a FrameOval using the extent.
                 adnRRect,            Do a FrameRoundRect using the extent.
                 adnShadow );         Draw drop-shadows against framed selections.
```

```
CntlAdornment   = SET OF adornPieces;
```

```
ControlCharSet  = SET OF Chr00..Chr1F;
```

Used by TEditText for allowable control chars.

Global Routines

```
PROCEDURE InitUDialog;
```

This routine initializes the UDialog unit. It must be called before any of the view classes in this unit can be created from 'view' resources.

```
FUNCTION  NewTemplateWindow (viewRsrcID: INTEGER;
      itsDocument: TDocument): TWindow;
```

This routine creates a new TWindow (or descendant) from the 'view' template with the given resource ID.

```
PROCEDURE RegisterType (typeName: Str255; protoObj: TObject);
```

This routine registers the given object (which can be an instance of any descendant of TObject) for use by TEvtHandler and its descendants when creating views from a template.

The TEntry Class

This class is used by TAssociation to form a very basic and cheap text item dictionary mechanism whose main use will be substitution of text in dialog-type window items. This class will be instantiated for each key string and its replacement that is desired in a dialog. A list of these entries is kept by a TAssociation. It is a simple subclass of TObject.
To duplicate the present function of ParamText, you'd use '^0', '^1', '^2', and '^3' for the first four key strings.

Fields

`fKey:`	`StringHandle;`	The handle to the target string.
`fValue:`	`StringHandle;`	The handle to the target's replacement string.

Initializing and Freeing Methods

```
PROCEDURE TEntry.IEntry (itsKey, itsValue: Str255);
```

This method creates strings using the Toolbox Utilities routine `NewString`.

```
PROCEDURE TEntry.Free; OVERRIDE;
```

This method frees associated data structures.

Debugging

```
PROCEDURE TEntry.Fields (PROCEDURE DoToField (fieldName: Str255;
                                              fieldAddr: Ptr;
                                              fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TEntriesList Class

This class is used by `TAssociation` to form a sorted list of entries. It is a subclass of `TSortedList`.

Comparison Method

```
FUNCTION TEntriesList.Compare (item1, item2: TObject): INTEGER; OVERRIDE;
```

This method compares two items in the list of entries.

The TAssociation Class

`TAssociation` manages a list of `TEntrys`. It is a subclass of `TObject`.

Fields

```
fEntries:       TEntriesList;
```

The list of entries.

Initializing and Freeing Methods

```
PROCEDURE TAssociation.IAssociation;
```

This method initializes the `fEntries` list.

```
PROCEDURE TAssociation.Free; OVERRIDE;
```

This method frees the `fEntries` list.

Accessing Methods

```
FUNCTION  TAssociation.ValueAt (keyStr: Str255; VAR valueStr: Str255): BOOLEAN;
```

This method returns the replacement string associated with the target string `keyStr`, or `""` (the null string) if there is no entry for `keyStr`.

```
FUNCTION  TAssociation.KeyAt (valueStr: Str255; VAR keyStr: Str255): BOOLEAN;
```

This method returns the target string associated with the replacement string `valueStr`, or `""` (the null string) if there is no entry for `valueStr`.

```
FUNCTION  TAssociation.EachEntryDo (PROCEDURE DoToEntry(theEntry: TEntry));
```

For each `TEntry` in `fEntries`, this method calls `DoToEntry`, passing the entry as an argument.

```
FUNCTION  TAssociation.EntryWithKey (keyStr: Str255): TEntry;
```

This method returns the first `TEntry` with the given target string.

```
FUNCTION  TAssociation.EntryWithValue (valueStr: Str255): TEntry;
```

This method returns the first `TEntry` with the given replacement string.

```
FUNCTION TAssociation.FirstEntryThat (FUNCTION TestEntry(theEntry: TEntry):
      BOOLEAN): TEntry;
```

This method returns the first `TEntry` that satisfies the given test function.

Methods for Adding/Removing Entries

```
PROCEDURE TAssociation.InsertEntry (keyStr, valueStr: Str255);
```

If `keyStr` exists in any of `fEntries`' target strings, this method replaces the associated replacement string with `valueStr`. If `keyStr` does not exist, this method adds a new `TEntry` to the end of the `fEntries` list.

```
PROCEDURE TAssociation.RemoveValueAt (keyStr: Str255);
```

This method removes the entry with the given key string.

```
PROCEDURE TAssociation.RemoveKeyAt (valueStr: Str255);
```

This method removes the first entry found with the given replacement string.

Debugging

```
PROCEDURE TAssociation.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                    fieldAddr: Ptr;
                                                    fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TDialogView Class

`TDialogView` implements modal dialog behavior. It acts as the topmost view in a template-driven dialog, and it undertakes several tasks mirroring the Dialog Manager. It is a subclass of `TView`.

Fields

`fDefaultItem:`	`IDType;`	The identifier of the default item.
`fCancelItem:`	`IDType;`	The identifier of the 'cancel' item.
`fParamTxt:`	`TAssociation;`	The list of TEntrys for replacement.
`fCurrentEditText:`	`TEditText;`	The currently selected TEditText item.
`fTEView:`	`TDialogTEView;`	Used for actual editing in TEditText views.
`fDismissed:`	`BOOLEAN;`	Indicates whether dialog has been dismissed.
`fDismisser:`	`IDType;`	The identifier of the view that caused the dismissal.

Initializing and Freeing Methods

```
PROCEDURE TDialogView.IDialogView (itsDocument: TDocument; itsSuperView: TView;
      itsLocation, itsSize: VPoint; itsHSizeDet, itsVSizeDet: SizeDeterminer;
      itsDefItemID, itsCancelItemID: IDType);
```

This method initializes allocated fields.

```
PROCEDURE TDialogView.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TDialogView.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TDialogView.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TDialogView.Free; OVERRIDE;
```

This method disposes of allocated fields.

Keyboard-Handling Methods

```
FUNCTION  TDialogView.DoKeyCommand (ch: Char; aKeyCode: INTEGER;
      VAR info: EventInfo): TCommand; OVERRIDE;
```

This method intercepts Tab and Return/Enter keystrokes, and handles them via the `HandleTab` and `HandleCR` methods.

```
FUNCTION  TDialogView.HandleCR (callingView: TView;
      VAR info: EventInfo): TCommand;
```

This method does whatever is associated with the Return and/or Enter keys. As a default, it calls

DismissDialog.

```
FUNCTION  TDialogView.HandleTab (callingView: TView;
      VAR info: EventInfo): TCommand;
```

This method does whatever is associated with the Tab key. As a default, it finds the next `TEditText` view (or the previous view, if the Shift key is down) and makes it the target view by calling `DoSelectEditText`.

```
FUNCTION  TDialogView.DoCommandKey (ch: Char; aKeyCode: INTEGER;
      VAR info: EventInfo): TCommand; OVERRIDE;
```

This method handles the command-'.' key sequence, which cancels the dialog.

Dialog Dismissal Methods

```
FUNCTION  TDialogView.CanDismiss (dismissing: IDType): BOOLEAN;
```

This method does any validation required before dismissing the dialog. `dismissing` is the identifier of the view which would cause the dismissal. If the dialog cannot yet be dismissed, `FALSE` is returned.

```
PROCEDURE TDialogView.CantDeselect (theEditText: TEditText);
```

This method is called if a `TEditText` item can't be deselected (usually as a result of failing validation).

```
PROCEDURE TDialogView.Close; OVERRIDE;
```

This method calls `CanDismiss` to assure dialog can be dismissed, and then calls `DismissDialog` if everything is OK.

```
PROCEDURE TDialogView.DismissDialog (dismisser: IDType);
```

This method performs whatever actions are necessary when the dialog is dismissed.

Global control functions

```
FUNCTION  TDialogView.DoChoice (origView: TView; itsChoice: INTEGER): TCommand;
      OVERRIDE;
```

This method handles the `mEditTextHit` message, which makes the selected `TEditText` object (represented by `origView`) the target view, and also handles messages from `TControls` which can dismiss dialogs.

```
FUNCTION  TDialogView.PoseModally: IDType;
```

This method is a distant cousin to the Toolbox `ModalDialog` call. It calls the main event loop repeatedly until the dialog is dismissed (the `fDismissed` field becomes `TRUE`), at which point `PoseModally` returns the identifier of the view that dismissed the dialog. This method will be called to simulate the current handling by the Toolbox for most modal dialogs.

TEditText Management Methods

```
PROCEDURE TDialogView.DoSelectEditText (theEditText: TEditText;
      selectChars: BOOLEAN);
```

This method attempts to make `theEditText` the selected edit text view. The `StopEdit` method for the current edit text view is called. Provided it returns true, `theEditText` is made the new current view. If `selectChars` is true then all of the characters in `theEditText` are selected.

```
PROCEDURE TDialogView.EachEditText (
      PROCEDURE DoToEditText(theEditText: TEditText));
```

This method applies the `DoToEditText` procedure to each `TEditText` descendant.

```
FUNCTION  TDialogView.MakeTEView: TDialogTEView;
```

This method creates the "floating" `TEView`, which is used to perform actual editting on a `TEditText` view (which is actually just a subclass of `TStaticText`).

```
PROCEDURE TDialogView.ParamTxt (keyStr, valueStr: Str255);
```

This method simply calls `fParamTxt.AtPut`.

```
PROCEDURE TDialogView.ReplaceText (theText: Handle);
```

This method calls `fParamTxt.EachEntryDo` using `Munger` to change all occurrences of target strings to their replacement values. Although this method can be called directly, it is usually only called by `TStaticText` views when their `Draw` methods are executed.

```
PROCEDURE TDialogView.SelectEditText (itsIdentifier: IDType;
      selectChars: BOOLEAN);
```

This method makes the given `TEditText` view the current view.

```
PROCEDURE TDialogView.SurveyEditText (VAR first, last, next,
      previous: TEditText);
```

This method looks through all subviews and finds the first, last, next, and previous `TEditText` fields. Called by `HandleTab` to determine options for tabbing.

Miscellaneous Methods

```
PROCEDURE TDialogView.GetDialogView: TView; OVERRIDE;
```

This method returns `SELF`. The result must be typed as `TDialogView` manually. (This was done so that it could be implemented as a `TView` method, without having to "include the world" unless necessary.)

Debugging

```
PROCEDURE TDialogView.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                   fieldAddr: Ptr;
                                                   fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TButton Class

This class implements a simple Control Manager button. It is a subclass of `TCtlMgr`.

Initializing and Freeing Methods

```
PROCEDURE TButton.IButton (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer; itsLabel: Str255);
```

This method initializes the button and installs it in the given superview. `fDefChoice` is set to `mButtonHit`.

```
PROCEDURE TButton.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TButton.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely

need to call it yourself.

```
PROCEDURE TButton.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

The TCheckBox Class

This class implements a simple Control Manager check box control. It is a subclass of `TCtlMgr`.

Initializing and Freeing Methods

```
PROCEDURE TCheckBox.ICheckBox (itsSuperView: TView;
      itsLocation, itsSize: VPoint; itsHSizeDet, itsVSizeDet: SizeDeterminer;
      itsLabel: Str255; isTurnedOn: BOOLEAN);
```

This method initializes a check box control and installs it in the given superview. `fDefChoice` is set to `mCheckBoxHit`. If `isTurnedOn` is `TRUE`, the control is initialized to be on.

```
PROCEDURE TCheckBox.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TCheckBox.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TCheckBox.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

Global Control Functions

```
PROCEDURE TCheckBox.DoChoice (origView: TView; itsChoice: INTEGER);
```

This method toggles the check box and sends the `mCheckBoxHit` message to its superview.

```
FUNCTION  TCheckBox.IsOn: BOOLEAN;
```

This method returns `TRUE` if the check box is currently on.

```
PROCEDURE TCheckBox.SetState (state, redraw: BOOLEAN);
```

This method turns the check box on or off, depending on the value of `state`. It redraws the check box if `redraw` is `TRUE`.

```
PROCEDURE TCheckBox.Toggle (redraw: BOOLEAN);
```

This method toggles the check box on or off. It redraws the check box if `redraw` is `TRUE`.

```
PROCEDURE TCheckBox.ToggleIf (matchState, redraw: BOOLEAN);
```

This method toggles the check box if the current state matches `matchState`. It redraws the check box if `redraw` is `TRUE`.

The TRadio Class

This class implements a simple Control Manager radio button control. It is a subclass of `TCtlMgr`.

Initializing and Freeing Methods

```
PROCEDURE TRadio.IRadio (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer; itsLabel: Str255;
      isTurnedOn: BOOLEAN);
```

This method initializes a radio button control and installs it in the given superview. `fDefChoice` is set to `mRadioHit`. If `isTurnedOn` is `TRUE`, the control is initialized to be on.

```
PROCEDURE TRadio.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TRadio.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TRadio.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

Global Control Functions

```
PROCEDURE TRadio.DoChoice (origView: TView; itsChoice: INTEGER);
```

This method toggles the radio button and sends the `mRadioHit` message to its superview.

```
FUNCTION TRadio.IsOn: BOOLEAN;
```

This method returns `TRUE` if the radio button is currently on.

```
PROCEDURE TRadio.SetState (state, redraw: BOOLEAN);
```

This method turns the radio button on or off, depending on the value of `state`. It redraws the check box if `redraw` is `TRUE`.

```
PROCEDURE TRadio.Toggle (redraw: BOOLEAN);
```

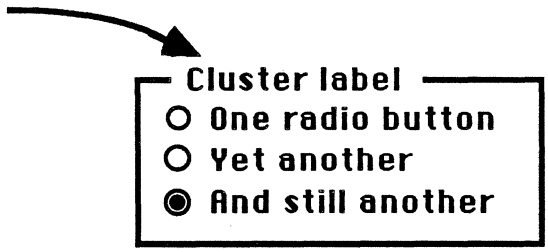
This method toggles the control on or off. It redraws the control if `redraw` is `TRUE`.

```
PROCEDURE TRadio.ToggleIf (matchState, redraw: BOOLEAN);
```

This method toggles the control if the current state matches `matchState`. It redraws the control if `redraw` is `TRUE`.

The TCluster Class

This class implements a *holding* view for radio buttons and/or other objects. It has two intrinsic functions: It understands an `mRadioHit` message from a subview, and can be used to contain other controls with a graphic label in the manner of Microsoft application dialogs, for example:



`TCluster` will usually be used to localize the handling of logically grouped controls. It is a simple subclass of `TControl`.

Fields

`fRsrcID:`	`INTEGER;`	The resource ID of the string list.
`fIndex:`	`INTEGER;`	The index of the label.
`fDataHandle:`	`StringHandle;`	The handle to the string data.

Initializing and Freeing Methods

```
PROCEDURE TCluster.ICluster (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer; itsRsrcID, itsIndex: INTEGER);
```

This method initializes the cluster and installs it in the given superview. `fDefChoice` is set to `mClusterHit`.

```
PROCEDURE TCluster.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TCluster.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TCluster.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TCluster.Free; OVERRIDE;
```

This method disposes of the handle.

Drawing/Focusing/Sizing/Mouse-Handling Methods

```
PROCEDURE TCluster.Draw (area: Rect); OVERRIDE;
```

This method's default action is to call the parent `DialogView`'s `ReplaceText` method (to replace wildcard strings), `TextBox` (if there is a label), and `FrameRect` (if a frame is desired).

Local Message-Handling Methods

```
PROCEDURE TCluster.DoChoice (origView: TView; itsChoice: INTEGER);  OVERRIDE;
```

If the message is `mRadioHit` and the radio button (`origView`) wasn't already selected, then this method turns the radio button on, and the remaining `TRadios` that are immediate descendants are turned off.

```
FUNCTION  TCluster.ReportCurrent: IDType;
```

This method returns the identifier of the currently selected radio button in the cluster.

Utility Methods

```
PROCEDURE TCluster.GetLabel (VAR theLabel: Str255);
```

This method returns the cluster's current label.

```
PROCEDURE TCluster.ReleaseLabel;
```

This method disposes of the label string handle.

```
PROCEDURE TCluster.SetLabel (theLabel: Str255; redraw: BOOLEAN);
```

This method sets the cluster label to the given string, forcing a redraw if requested.

Debugging

```
PROCEDURE TCluster.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                fieldAddr: Ptr;
                                                fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TIcon Class

This class implements an icon item that can serve as a crude form of button if enabled. It is a simple subclass of `TControl`.

Fields

`fPreferColor:`	`BOOLEAN;`	TRUE if `'cicn'` resources should be checked before `'ICON'` resources.
`fRsrcID:`	`INTEGER;`	The resource ID of the icon.
`fDataHandle:`	`Handle;`	The handle to the icon data.

Initializing and Freeing Methods

```
PROCEDURE TIcon.IIcon (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer; itsRsrcID: INTEGER; preferColor:
      BOOLEAN);
```

This method initializes an icon item and installs it in the given superview. If `preferColor` is `TRUE`, then the icon will be fetched from a `'cicn'` resource, instead of from an `'ICON'` if `preferColor` is `FALSE`. `fDefChoice` is set to `mIconHit`.

```
PROCEDURE TIcon.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TIcon.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method, and is included for completeness. You will rarely need to call it yourself.

```
PROCEDURE TIcon.WriteRes (theResource: ViewRsrcHndl;
    VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TIcon.Free; OVERRIDE;
```

This method disposes of the handle.

Drawing/Focusing/Sizing/Mouse-Handling

```
PROCEDURE TIcon.Draw (area: Rect); OVERRIDE;
```

This method's default action is to call `PlotIcon` or `PlotCIcon`.

Utility Methods

```
PROCEDURE TIcon.ReleaseIcon;
```

This method disposes of the icon data. (Actually, "forgets" about resource data location.)

```
PROCEDURE TIcon.SetIcon (theIcon: Handle; redraw: BOOLEAN);
```

This method sets the `fDataHandle` field to the given handle, forcing a redraw if requested.

Debugging

```
PROCEDURE TIcon.Fields (PROCEDURE DoToField (fieldName: Str255;
                                             fieldAddr: Ptr;
                                             fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TPattern Class

This class implements a pattern item that can serve as a crude form of button if enabled. It is a subclass of `TControl`.

Fields

`fPreferColor:`	`BOOLEAN;`	TRUE if `'ppat'` resources should be checked before `'PAT '` resources.
`fRsrcID:`	`INTEGER;`	The resource ID of the pattern.
`fDataHandle:`	`Handle;`	The handle to the pattern data.

Initializing and Freeing Methods

```
PROCEDURE TPattern.IPattern (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer; itsRsrcID: INTEGER; preferColor:
      BOOLEAN);
```

This method initializes a pattern item and installs it in the given superview. If `preferColor` is `TRUE`, then the pattern will be fetched from a `'ppat'` resource, instead of from a `'PAT '` resource if `preferColor` is `FALSE`. `fDefChoice` is set to `mPatternHit`.

```
PROCEDURE TPattern.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TPattern.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TPattern.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TPattern.Free; OVERRIDE;
```

This method disposes of the handle.

Drawing/Focusing/Sizing/Mouse-Handling

```
PROCEDURE TPattern.Draw (area: Rect); OVERRIDE;
```

This method's default action is to call `FillRect` or `FillCRect`.

Utility Methods

```
PROCEDURE TPattern.ReleasePattern;
```

This method disposes of the pattern data. (Actually, "forgets" about resource data location.)

```
PROCEDURE TPattern.SetPattern (thePattern: Handle; redraw: BOOLEAN);
```

This method sets the `fDataHandle` field to the given handle, forcing a redraw if requested.

Debugging

```
PROCEDURE TPattern.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                fieldAddr: Ptr;
                                                fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TPicture Class

This class implements a picture item that can serve as a crude form of button if enabled. It is a simple subclass of `TControl`.

Fields

`fRsrcID:`	`INTEGER;`	The resource ID of the picture.
`fDataHandle:`	`PicHandle;`	The handle to the picture data.

Initializing and Freeing Methods

```
PROCEDURE TPicture.IPicture (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer; itsRsrcID: INTEGER);
```

This method initializes a picture item and installs it in the given superview. `fDefChoice` is set to `mPictureHit`.

```
PROCEDURE TPicture.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TPicture.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included for completeness. You will rarely need to call it yourself.

```
PROCEDURE TPicture.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TPicture.Free; OVERRIDE;
```

This method disposes of the handle.

Drawing/Focusing/Sizing/Mouse-Handling

```
PROCEDURE TPicture.Draw (area: Rect); OVERRIDE;
```

This method's default action is to call `DrawPicture`.

Utility Methods

```
PROCEDURE TPicture.ReleasePicture;
```

This method disposes of the picture data. (Actually, "forgets" about resource data location.)

```
PROCEDURE TPicture.SetPicture (thePicture: PicHandle; redraw: BOOLEAN);
```

This method sets the `fDataHandle` field to the given handle, redrawing if requested.

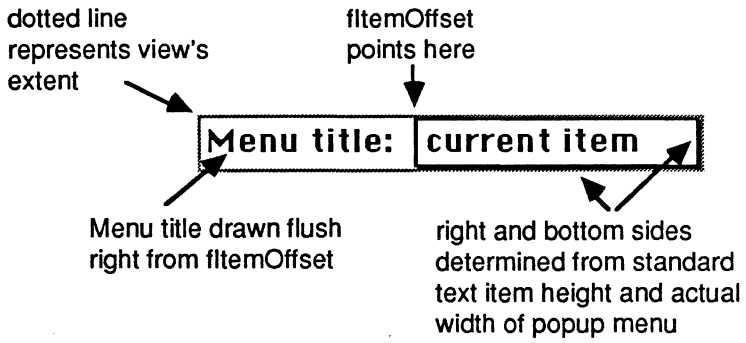
Debugging

```
PROCEDURE TPicture.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                fieldAddr: Ptr;
                                                fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TPopup Class

This class implements a simple pop-up menu selector, following the guidelines for pop-up menus laid down by the Apple Human Interface group. See Inside Macintosh Vol. V, pages V-241 and V-242, for an explanation.



`TPopup` is a simple subclass of `TControl`.

Fields

`fRsrcID:`	`INTEGER;`	The resource ID of the `'MENU'` resource.
`fMenuID:`	`INTEGER;`	The menu ID of the popup menu.
`fMenuHandle:`	`MenuHandle;`	The menu handle.
`fCurrentItem:`	`INTEGER;`	The currently selected menu item.
`fItemOffset:`	`INTEGER;`	The offset from the left edge of the extent to the pop-up selector.

Initializing and Freeing Methods

```
PROCEDURE TPopup.IPopup (itsSuperView: TView; itsLocation, itsSize: VPoint;
      itsHSizeDet, itsVSizeDet: SizeDeterminer;
      itsRsrcID, itsCurrentItem, itsItemOffset: INTEGER);
```

This method initializes a pop-up menu selector and installs it in the given superview. The title is drawn flush right starting at `fItemOffset`. The pop-up selector is drawn to the right of `fItemOffset`, the width and height being determined from the dimensions of the pop-up menu. `fDefChoice` is set to `mPopupHit`.

```
PROCEDURE TPopup.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TPopup.WRes (theResource: ViewRsrcHndl; VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included for completeness. You will rarely need to call it yourself.

```
PROCEDURE TPopup.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TPopup.Free; OVERRIDE;
```

This method disposes of the handle.

Drawing/Focusing/Sizing/Mouse-Handling Methods

```
PROCEDURE TPopup.AdjustBotRight;
```

This is a utility method to determine the bottom and right sides of the view, based on the menu information.

```
PROCEDURE TPopup.CalcLabelRect (VAR theRect: Rect);
```

This method calculates the rectangle for the pop-up menu label.

```
PROCEDURE TPopup.CalcMenuRect (VAR theRect: Rect);
```

This method calculates the rectangle for the pop-up menu itself (that is, the menu minus the title string.)

```
PROCEDURE TPopup.Draw (area: Rect); OVERRIDE;
```

This method's default action is to draw the menu title, frame a rectangle to its immediate right, shadow it, and call `TextBox` to draw the menu selector's text.

```
PROCEDURE TPopup.DrawLabel (area: Rect);
```

This method draws the popup menu's label.

```
PROCEDURE TPopup.DrawPopup (area: Rect);
```

This method draws the popup menu itself (minus the label).

```
PROCEDURE TPopup.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
      VAR hysteresis: Point): TCommand; OVERRIDE;
```

This method's default action is to invert the menu title, call `PopUpMenuSelect`, and set `fCurrentItem` to the result.

Utility Methods

```
FUNCTION  TPopup.GetCurrentItem: INTEGER;
```

This method returns the number of the currently selected menu item.

```
PROCEDURE TPopup.GetItemText (item: INTEGER; VAR theText: Str255);
```

This method returns the text of the given item in the menu.

```
PROCEDURE TPopup.ReleasePopup;
```

This method disposes of the menu data.

```
PROCEDURE TPopup.SetCurrentItem (item: INTEGER; redraw: BOOLEAN);
```

This method sets the number of the currently selected menu item, redrawing if requested.

```
PROCEDURE TPopup.SetPopup (theMenu: MenuHandle; theRsrcID, currentItem: INTEGER;
      redraw: BOOLEAN);
```

This method sets the `fMenuHandle` field to the given handle, sets the currently selected item, and redraws if requested.

Debugging

```
PROCEDURE TPopup.Fields (PROCEDURE DoToField (fieldName: Str255;
                                              fieldAddr: Ptr;
                                              fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TStaticText Class

This class implements a static text item that can serve as a crude form of button if enabled. The text cannot be editted. It is a simple subclass of `TControl`.

Fields

`fRsrcID:`	`INTEGER;`	The resource ID of the string list.
`fIndex:`	`INTEGER;`	The index of the string.
`fJust:INTEGER;`	The string's justification.	
`fDataHandle:`	`Handle;`	The handle to the string data.

Initializing and Freeing Methods

```
PROCEDURE TStaticText.IStaticText (itsSuperView: TView;
      itsLocation, itsSize: VPoint; itsHSizeDet, itsVSizeDet: SizeDeterminer;
      itsRsrcID, itsIndex: INTEGER);
```

This method initializes a static text item and installs it in the given superview. `fDefChoice` is set to `mStaticTextHit`.

```
PROCEDURE TStaticText.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TStaticText.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TStaticText.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TStaticText.Free; OVERRIDE;
```

This method disposes of the handle.

Drawing/Focusing/Sizing/Mouse-Handling

```
PROCEDURE TStaticText.Draw (area: Rect); OVERRIDE;
```

This method's default action is to call `TextBox`.

```
FUNCTION  TStaticText.DoSubstitution (VAR realText: Handle): BOOLEAN;
```

Performs text subsitutions on the view's text each time the view is drawn. Returns true if no error occurs.

```
PROCEDURE TStaticText.ImageText (text: ptr; length: LONGINT; box: Rect;
      just: INTEGER): BOOLEAN;
```

Called from the `Draw` method to actually draw the text. Used the `TextEdit` routine `TextBox` to draw the text.

Utility Methods

```
PROCEDURE TStaticText.GetText (VAR theText: Str255);
```

This method fetches the current text, truncating to 255 characters if necessary.

```
PROCEDURE TStaticText.ReleaseText;
```

This method disposes of the string data. (Actually, "forgets" about resource data location.)

```
PROCEDURE TStaticText.SetJustification (theJust: INTEGER; redraw: BOOLEAN);
```

This method sets the text's justification, forcing a redraw if requested.

```
PROCEDURE TStaticText.SetText (theText: Str255; redraw: BOOLEAN);
```

This method sets the text to the given string, forcing a redraw if requested.

Debugging

```
PROCEDURE TStaticText.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                   fieldAddr: Ptr;
                                                   fieldType: INTEGER)); OVERRIDE;
```

This method implements debugging support for the inspector.

The TDialogTEView Class

This class is a subclass of `TEView` that `DialogView` uses to perform actual editing on `TEditText` views.

Fields

`fEditText:`	`TEditText;`	The associated `TEditText` object.

Drawing/Focusing/Sizing/Mouse-Handling

```
PROCEDURE TDialogTEView.DrawContents; OVERRIDE;
```

This method is overridden to do nothing. (It relies on the `TEditText` view to draw the actual item.)

TEditText Management Methods

```
PROCEDURE TDialogTEView.InstallEditText (theEditText: TEditText;
      selectChars: BOOLEAN);
```

This method initializes the object based on the given `TEditText` view.

```
PROCEDURE TDialogTEView.InstallSelection (wasActive, beActive: BOOLEAN);
      OVERRIDE;
```

This method is overridden to clean up auto-scrolling.

```
PROCEDURE TDialogTEView.RecalcText; OVERRIDE;
```

This method is overridden to optimize recalculation of the text record.

```
PROCEDURE TDialogTEView.ScrollSelectionIntoView; OVERRIDE;
```

This method is overridden to optimize scrolling of the record.

Debugging

```
PROCEDURE TDialogTEView.Fields (PROCEDURE DoToField (fieldName: Str255;
      fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
```

This method implements debugging support for the inspector.

The TEditText Class

This class implements a simple editable text item. `TEditText` is implemented as a subclass of `TStaticText`. When the item needs to be edited, the parent `DialogView` places a "floating" `TEView` on top of the view.

Fields

`fControlChars:`	`ControlCharSet;`	Control characters that are allowed and not passed on.
`fMaxChars:`	`INTEGER;`	Maximum number of characters to accept.
`fTEView:`	`TTEView;`	The associated `TTEView` object.

Initializing and Freeing Methods

```
PROCEDURE TEditText.IEditText (itsSuperView: TView;
      itsLocation, itsSize: VPoint; itsMaxChars: INTEGER);
```

This method initializes an editable text item and installs it in the given superview. The record is created without styles, auto-wrap is turned on, and the view defaults to `sizeFixed` for its horizontal determiner and `sizeVariable` for its vertical determiner. The viewable area of the text record is inset by a few pixels.

```
PROCEDURE TEditText.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TEditText.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included for completeness. You will rarely need to call it yourself.

```
PROCEDURE TEditText.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

Drawing/Focusing/Sizing/Mouse-Handling Methods

```
PROCEDURE TEditText.DoIdle (phase: IdlePhase); OVERRIDE;
```

This method handles caret blinking.

```
FUNCTION  TEditText.DoKeyCommand (ch: Char; aKeyCode: INTEGER;
      VAR info: EventInfo): TCommand; OVERRIDE;
```

This method acts on Tab and Return/Enter keystrokes based on the state of `fCRAccept` and `fTabAccept`.

```
FUNCTION  TEditText.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
      VAR hysteresis: Point): TCommand; OVERRIDE;
```

This method either selects the `TEditText` item if it wasn't active, or passes its click along to the floating `TTEView`.

```
FUNCTION  TEditText.DoMenuCommand (aCmdNumber: INTEGER): TCommand; OVERRIDE;
```

This method handles menu commands if the edit text is currently selected by calling the `TEView`'s `DoMenuCommand` method.

```
FUNCTION  TEditText.DoSetupMenus; OVERRIDE;
```

This method sets menu items as appropriate for editable text.

```
PROCEDURE TEditText.Draw (area: Rect); OVERRIDE;
```

If the text item is presently being editted, this method calls the `TEView`'s `Draw` method, otherwise calls `INHERITED Draw`, which draws the view using `TextBox`.

```
FUNCTION  TEditText.DoSubstitution (VAR realText: Handle): BOOLEAN; OVERRIDE;
```

Overridden to avoid performing substitutions on the view's text.

```
PROCEDURE TEditText.ImageText (text: ptr; length: LONGINT; box: Rect;
      just: INTEGER): BOOLEAN;
```

Called from the `Draw` method to actually draw the text.

```
PROCEDURE TEditText.Locate (h, v: VCoordinate; invalidate: BOOLEAN); OVERRIDE;
```

This method locates the `TEditText` object, and then forces the `TDialogTEView` to locate over it.

```
PROCEDURE TEditText.Resize (width, height: VCoordinate;
      invalidate: BOOLEAN); OVERRIDE;
```

This method resizes the `TEditText` object and the associated `TDialogTEView`.

Utility Functions

```
PROCEDURE TEditText.GetText (VAR theText: Str255); OVERRIDE;
```

This method returns the current text, fetching it from the floating `TEView` if appropriate.

```
PROCEDURE TEditText.InstallSelection (wasActive, beActive: BOOLEAN); OVERRIDE;
```

This method calls `TDialogTEView`'s `InstallSelection` method if appropriate.

```
PROCEDURE TEditText.RestartEdit;
```

This method focuses and selects the entire `TEView` record.

```
PROCEDURE TEditText.SetSelection (selStart, selEnd: INTEGER;
      redraw: BOOLEAN); OVERRIDE;
```

This method sets the selection range in the item.

```
PROCEDURE TEditText.SetText (theText: Str255; redraw: BOOLEAN); OVERRIDE;
```

This method is a convenience method for setting the control's text by directly passing a string, as opposed to having to create a `StringHandle`.

```
PROCEDURE TEditText.SetJustification (theJust: INTEGER; redraw: BOOLEAN);
      OVERRIDE;
```

This method sets the justification of the view's text.

```
PROCEDURE TEditText.StartEdit (selectChars: BOOLEAN; theTEView: TDialogTEView);
```

This method installs the present text into the floating `TEView`.

```
FUNCTION  TEditText.StopEdit: BOOLEAN;
```

This method fetches the text from the floating `TEView` and validates its text, returning the result.

```
FUNCTION  TEditText.Validate: BOOLEAN; OVERRIDE;
```

This method returns `TRUE` if the text is valid. It returns `TRUE` as a default.

Debugging

```
PROCEDURE TEditText.Fields (PROCEDURE DoToField (fieldName: Str255;
                                                 fieldAddr: Ptr;
                                                 fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

The TNumberText Class

This class implements a simple edittable text item that only accepts long integer numbers within a given range. `TNumberText` is a simple subclass of `TEditText`.

Fields

`fMinimum:`	`LONGINT;`	Minimum value, used to validate the view.
`fMaximum:`	`LONGINT;`	Maximum value, used to validate the view.

Initializing and Freeing Methods

```
PROCEDURE TNumberText.INumberText (itsSuperView: TView;
      itsLocation, itsSize: VPoint; itsValue, itsMinimum, itsMaximum: INTEGER);
```

This method initializes the object to the given values.

```
PROCEDURE TNumberText.IRes (itsDocument: TDocument; itsSuperView: TView;
      VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the object from a `'view'` resource.

```
PROCEDURE TNumberText.WRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method,and is included for completeness. You will rarely need to call it yourself.

```
PROCEDURE TNumberText.WriteRes (theResource: ViewRsrcHndl;
      VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

Utility Functions

```
FUNCTION  TNumberText.GetValue: LONGINT;
```

This method returns the value of the text.

```
PROCEDURE TNumberText.SetValue (newValue: LONGINT; redraw: BOOLEAN);
```

This method sets the text to the given value, redrawing if requested.

```
FUNCTION  TNumberText.Validate: BOOLEAN; OVERRIDE;
```

This method returns `TRUE` if the number is within the range `fMinimum` to `fMaximum`, inclusive.

Debugging

```
PROCEDURE TNumberText.Fields (PROCEDURE DoToField (fieldName: Str255; fieldAddr:
      Ptr; fieldType: INTEGER)); OVERRIDE;
```

This method provides debugging support for the inspector.

Format of the 'view' resource

View components are listed hierarchically, with ancestral objects' parameters listed first. For the sake of clarity, the resource description is given in pseudo-Pascal format which is more descriptive than informative. The file `ViewTypes.r` in the folder `'MacApp Resource Files:'` contains the actual resource description of `'view'` resources.

Font names, when required, can default to null strings (`''`) for the system font, and to a single character A (`'A'`) for the application font.

```
ViewResource =    RECORD
      numViews:   INTEGER;                          Number of subviews
      theViews:   ARRAY [1..numViews] OF ViewInfo;  View descriptions
END;

ViewTemplate = PACKED RECORD
      itsParentID:          IDType;                 Identifier of superview
      thisViewID:           IDType;                 This view's identifier
      itsLocation:          VPoint;                 Location inside parent view
      itsSize:              VPoint;                 Size of this view
      itsVSizeDet:          sizeSuperView..sizeFixed; Vertical size determiner
      itsHSizeDet :         sizeSuperView..sizeFixed; Horizontal size determiner
      isShown:              BOOLEAN;                Whether this view is shown
      isEnabled:            BOOLEAN;                Whether this view is enabled
      (filler:              0..255;)
      itsSignature:         IDType;                 Template signature
      itsType:              Str255;                 Class name (actually variable size)
      itsData:              ARRAY [0..0] OF INTEGER; Extra view-specific data (detailed below)
END;


WindowTemplate = PACKED RECORD
      procID:               INTEGER;                Window procID
      hasGoAway:            BOOLEAN;                Whether window has go-away box
      resizable:            BOOLEAN;                Whether window is resizable
      isModal:              BOOLEAN;                Whether window is modal
      doFirstClick:         BOOLEAN;                Whether first click in window should be handled
      freeOnClosing:        BOOLEAN;                Whether window object is freed on closing
      disposeOnFree:        BOOLEAN;                Whether window is disposed of on closing
      closesDocument:       BOOLEAN;                Whether closing window closes document
      openInitially:        BOOLEAN;                Whether window is opened at document creation
      mustAdaptToScreen:    BOOLEAN;                Whether window is adapted to fit screen
      stagger:              BOOLEAN;                Whether window is staggered
      mustForceOnScreen:    BOOLEAN;                Whether window should be forced on screen
      vertCenter:           BOOLEAN;                Whether window is vertically centered
      horzCenter:           BOOLEAN;                Whether window is horizontally centered
      (filler:              0..7;)
      targetID:             IDType;                 Identifier of fTarget'ed view
      title:                Str255;                 Window title (actually variable size)
END;
```

```
ScrollerTemplate = RECORD
      wantVSBar:          BOOLEAN;      Whether vertical scroll bar wanted
      wantHSBar:          BOOLEAN;      Whether horizontal scroll bar wanted
      vertMax:            LONGINT;      Maximum vertical value
      horzMax:            LONGINT;      Maximum horizontal value
      vScrollUnits:       INTEGER;      Number of units to scroll vertically
      hScrollUnits:       INTEGER;      Number of units to scroll horizontally
      vConstrain:         BOOLEAN;      Whether vertical scroll bar is constrained
      hConstrain:         BOOLEAN;      Whether horizontal scroll bar is constrained
      sBarOffsets:        Rect;         Offsets for scroll bars during resizing
END;

DialogViewTemplate =      RECORD
      defaultID:          IDType;       Identifier of default item
      cancelID:           IDType;       Identifier of 'cancel' item
END;

ControlTemplate = PACKED RECORD
      itsAdornment:       CntlAdornment;  Adornment flags
      (filler1:           0..255;)
      itsPenSize:         Point;        Pen size
      isSizable:          BOOLEAN;      Whether control is sizable (regardless of view)
      isDimmed:           BOOLEAN;      Whether control is dimmed
      isHilited:          BOOLEAN;      Whether control is hilited
      canDismiss:         BOOLEAN;      Whether control can dismiss dialog
      (filler2:           0..4095;)
      itsInset:           Rect;         Control's inset from edges of view
      itsTextFace:        Style;        Control's text face
      itsTextSize:        INTEGER;      Control's text size
      itsTextColor:       RGBColor;     Control's color
      itsFontName:        Str255;       Name of font (actually variable size)
END;

ButtonTemplate =  PACKED RECORD
      itsLabel:           Str255;       Button's label (actually variable size)
END;

CheckBoxTemplate = RECORD
      isOn:               BOOLEAN;      Whether check box is initially "on"
      itsLabel:           Str255;       Check box's label (actually variable size)
END;

RadioTemplate =   RECORD
      isOn:               BOOLEAN;      Whether radio button is initially "on"
      itsLabel:           Str255;       Radio button's label (actually variable size)
END;
```

```
ScrollBarTemplate = RECORD
      cntlData:            TControlData;          TControl initialization
      direction:           VHSelect;              Scroll bar's direction
      itsValue:            LONGINT;               Initial value
      itsMinimum:          LONGINT;               Minimum value
      itsMaximum:          LONGINT;               Maximum value
END;

ClusterTemplate = RECORD
      itsLabel:            Str255;                Cluster's label (actually variable size)
END;

IconTemplate =    RECORD
      preferColor:BOOLEAN; Whether color icon is preferred
      rsrcID:              INTEGER;               Resource ID of 'cicn' or 'ICON' resource
END;

PatternTemplate = RECORD
      preferColor:BOOLEAN; Whether color pattern is preferred
      rsrcID:              INTEGER;               Resource ID of 'ppat' or 'PAT ' resource
END;

PictureTemplate = RECORD
      rsrcID:              INTEGER;               Resource ID of 'PICT' resource
END;

PopupTemplate =   RECORD
      rsrcID:              INTEGER;               Resource ID of 'MENU' resource
      currentItem:         INTEGER;               Initially selected item number
      itemOffset:          INTEGER;               Offset from left of view to pop-up control
END;

StaticTextTemplate = RECORD
      just:                INTEGER;               Justification for text
      data:                Str255;                Initial text (actually variable size)
END;

EditTextTemplate = RECORD
      maxChars:            INTEGER;               Maximum number of characters accepted
      controlChars:        ControlCharSet;        Valid control characters
END;

NumberTextTemplate = RECORD
      itsValue:            LONGINT;               Initial value
      itsMinimum:          LONGINT;               Minimum value
      itsMaximum:          LONGINT;               Maximum value
END;
```

```
TEViewTemplate =  RECORD
      hasStyle:            BOOLEAN;          Whether record is styled
      autoWraps:           BOOLEAN;          Whether record auto-wraps characters
      acceptsChanges:      BOOLEAN;          Whether record accepts changes
      freesText:           BOOLEAN;          Whether object automatically frees text
      (filler:             0..4095;)
      keyCmdNumber:        INTEGER;          Command number for typing characters
      maxChars:            INTEGER;          Maximum number of characters allowed
      scrollerID:          IDType;           ID of scroller
      inset:               Rect;             Insets of viewRect from edges of view
      just:                INTEGER;          Record justification
      itsTextFace:         Style;            Record's text face
      itsTextSize:         INTEGER;          Record's text size
      itsTextColor:        RGBColor;         Record's color
      itsFontName:         Str255;           Name of font (actually variable size)
END;
```