eXpress Dialog Form Designer

4 March 2010

Table of Contents

Dialog Forms

Using Dialog Forms

Dialog forms provide a general-purpose dialog window to eXpress products. Dialog forms are available, using eXpress Scripting.

Dialog Forms are implemented as the TDialogForm class.

Using

The following sequence of actions is used to show and access results of a Dialog Form.

1. Create the dialog form object using the New or Create statement.
2. Load the dialog form file its binary form (BFM) file using the LoadForm method.
3. Optional initialize Global variables and/or controls on the Dialog Form.
4. Show the Dialog Form using the ShowForm method. The ShowForm method retruns a ModalResult value that may be used in the calling script.
5. Access results of the Dialog Form. This happens only after the Dialog Form is closed by useing the Close method in the Dialog Form's event action script, or by setting the ModalResult to a value > 0.
6. Free the Dialog Form using the Delete statement (see BasicScript Language Elements, PascalScipt Language Elements or JScript Language Elements) or the Free method.

Dialog Forms Scope

Dialog Forms and their associated Event Action Scripts run outside the environment of script that calls them — meaning the Dialog Form's script cannot directly access data in the caller's environment. A mechanism for exchanging data between the Dialog Form and its caller is provided with the GetSessionVar and SetSessionVar methods of the TTermScreen Object.

Dialog Forms can also return a ModalResult. This return is done using the by setting the ModalResult variable with the Dialog Form's event action script.

For samples of dialog forms, see the .ACT and .BFM files in the installed Scripts directory.

Dialog Form Designer

Use the Dialog Form Designer toolbar and the companion dialogs (Dialog Form, Property Editor and Dialog Form Action Script Editor) to alter the appearance of the basic terminal screen (as initially captured or converted), add new controls and automate end-user interface functions.

General Features

The Dialog Form Designer is very similar in appearance to the development environment of several object-oriented visual languages. The Dialog Form is the canvas upon which you place Windows controls (buttons, list boxes, etc.). The Dialog Form Designer toolbar is the pallet from which you select the controls that you want to place on the form. Each control has its own unique set of Properties, dependent upon the type of control selected, that allow you to specify how the control will appear and behave on the form.

The Dialog Form

The initial design form contains a menu bar at the top that has three disabled (grayed out) menu names. These menus are enabled at runtime and used to access standard options (e.g., Print Setup) at runtime. You may add other menus specific to the form and they will be displayed here, but they will not cause any actions to be invoked during the design phase.

The dialog form is a Windows control, and like any Windows control, has a set of properties associated with that control. For properties and event actions, see Form Properties.

The Dialog Form Designer Tool Bar

The tool bar (initially at the top of the monitor viewing area) can be moved anywhere on the window that is convenient. To move, first select Float Tool Bar from the tool bar’s View menu. Next, place the mouse cursor over the caption bar (over the words, "Dialog Form Designer"). While holding down the left mouse button, drag the toolbar to the desired location.

Adding Controls and Fields

To add controls to the Dialog Form, click the appropriate control button on the tool bar. Note: Adding menu controls are done by selecting Menu Designer from the Tools menu or clicking the corresponding button on the tool bar.

With a control button selected on the toolbar, move the mouse cursor over an empty area of the form window and click the left mouse button. The new control will be placed on the form where you clicked. Alternately, you may hold down the left mouse button and drag the mouse to create the initial size and location of the control. A box will grow and shrink as you drag in any direction. When the left mouse button is released, the control will appear, already selected and ready to move or edit.

To edit, click the left mouse button over the control and use the Property Editor window to change the desired property.

Moving and Sizing Controls

Dialog controls may be moved and resized with the mouse. Select the desired control by moving the mouse cursor over the control and clicking the left mouse button. When selected, the control will have sizing handles around it.

To size, move the mouse cursor over one of the sizing handles (the mouse cursor will change to sizing arrows), then press and hold the left mouse button and drag the sizing handle in the desired direction. When the mouse button is released, the control will snap to the new size.

To change the location on the form, move the mouse cursor into the center area of the control and press and hold the left mouse button. With the mouse, drag and drop the control to the desired location.

Aligning and Sizing Controls

The alignment and sizing feature allows any group of controls to be automatically aligned with or sized to a "base" control (any single Dialog control). The process is made simple by first selecting the base control with the left mouse button. Next, while holding down the Shift key, other controls to be aligned or sized are selected with the left mouse button (notice that the sizing handles become gray). Finally, right click on any of the selected controls and choose an alignment or sizing option from the popup menu.

Alternately, you may select a group of controls by moving the mouse cursor to a point on the dialog form away from any control (the gray part). While holding the left mouse button, drag the mouse to encompass the controls you want to align or size. Note: The last control selected becomes the base control.

Alignment and sizing are based on the original alignment and size of the base control. Alignment allows vertical alignment to be left justified, centered or right justified to the base control; horizontal alignment to be aligned across the top, middle or bottom of the base control. Sizing changes the width or height of the selected controls to match that of the base control.

Automatic Properties Assignment

Much like the ability to automatically size and align groups of controls (see above), controls may have their properties (font, color, etc.) set as a group. The procedure is practically the same. While holding down the left mouse button, drag the mouse over the controls whose properties are to change. Next, change the property or properties desired on the Property Editor window.

Most controls have "Parent" properties (ParentCtl3D, ParentFont and/or ParentShowHint). If a parent property is set to True, then the control takes on the property of the parent control. Currently, there are only two types of parent controls: the form itself and the panel control. For example, if a Button on the form has the ParentFont property set to True, the Button caption will use the same font assigned to the form. Note: Changing the Font property on the Button will automatically set ParentFont property to False.

A Panel can have controls placed on it so that whenever you move the panel during the design phase, all the controls on the panel move with it. Likewise, if an action disables the Panel at runtime, all the controls on the Panel are also disabled since the Panel is Parent to all the controls placed on the panel.

Automatic Alignment to Client Area

Some controls (panels, list boxes, text labels, etc.) can be aligned to all or part of the client area of a parent control. The Align property allows the automatic alignment of the control to the top ("alTop"), bottom (alBottom"), left (alLeft") or right ("alRight") edge of the parent control. If the Align property is set to "alNone" (the default), the control remains wherever it is placed on the parent control. A sixth setting, "alClient", can be used to align the control within the client area of the control. If multiple controls are aligned on the parent control, they are aligned in the precedence they were placed on the control.

An example of using aligned controls would be when you wanted to place a splitter between two list boxes thus giving the user the ability to determine how much of the list boxes would be visible at a given time.

File menu

New Form

Use this selection to close the current dialog form and start a new form.

Open

Use this selection to open an existing form.

Save

Use this selection to save alterations to the current form.

Save As

Use this selection to save the current form to a different binary form file.

Set Default Form/Script Folder

Use this option to establish a default form/script folder. When the default form/script folder is set, the next file Open or Save As will default to that folder.

Edit menu

Cut (Ctrl+X)

Use this selection to cut the selected control(s) to the Windows clipboard.

Copy (Ctrl+C)

Use this selection to copy the selected control(s) to the Windows clipboard.

Paste (Ctrl+V)

Use this selection to paste the contents of the Windows clipboard to the form.

Delete (Del)

Use this selection to delete the selected control(s).

Delete All Controls

Use this selection to clear the form.

Cut to File

Use this selection to cut the selected control(s) to a file (.clp).

Copy to File

Use this selection to cut the selected control(s) to a file (.clp).

Paste from File

Use this selection to paste the contents a file to the form.

Select All

Use this command to select all controls on the form.

Form Edits...

Use this selection to display the alignment, sizing, tab order and grid options popup menu. This selection is the same as doing a right mouse click over a selected control.

Options menu

Always save before Execute

When this option is checked, the form will be saved automatically before the Run (Show) Form command is performed.

Hide designer windows on Execute

With this option is set, the designer's windows will be hidden when the Run (Show) Form command is performed.

View menu

Tool Bar (F6)

Use this selection to return emphasis to the Dialog Form Designer toolbar.

Properties Window (F7)

Use this selection to open the Property Editor window.

Design Window (F8)

Use this selection to return emphasis to the Dialog Form window from the Dialog Form Designer toolbar.

Action Edit Window (F9)

Use this selection to open the Dialog Form Action Script Editor.

Float Tool Bar

Use this selection to unlock the toolbar. This selection allows you to drag the tool bar by holding down the left mouse button over the toolbar caption and dragging it to another location.

Help menu

Contents

This selection starts the Windows Help program and displays all help topics for the Dialog Form Designer.

About

Use this selection to display version and copyright notification.

The Designer Toolbar

Immediately below the menu bar is the Designer Toolbar. This toolbar contains redundant controls for most of the selections available through menu selections. In addition, there are two buttons that allow you to view/execute the finished form as it would appear at runtime:

Execute (Show) Form

Use this control to display the finished form.

Execute (Show) Form with Debugging

Use this button to step through the event action code while displaying the form.

The Designer buttons

The buttons along the bottom of the Dialog Form Designer toolbar are used to select the type of control you want to place on the form.

Pointer button

Use this selection to cancel adding a control, thus returning the mouse to a pointer tool. Use the pointer tool to select control(s) on the form.

Text Label

Use this selection to add a text label to a parent control (form or panel). This control is useful to label controls that have no caption property (e.g., Edit Boxes).

For properties, see Text Labels.

Edit Box

Use this selection to add an edit box to a parent control.

For properties, see Edit Boxes.

Command Button

Use this selection to add a command button to a parent control (e.g., OK button, Cancel button, Query button, etc.).

For properties, see Command Buttons.

Speed Button

Use this selection to add a speed button to a parent control. A speed button is a non-windowed control meaning it takes less system resources. Since it is a non-windowed control, you may not tab to it.

You can make a group of speed buttons act like radio buttons by setting the group index to something other than 0.

Speed buttons can also have that flat property that makes the outline show on mouse fly-over.

For properties, see Speed Buttons.

Check Box

Use this selection to add a check box to a parent control. Use check boxes when multiple options may be selected by the user.

For properties, see Check Boxes.

 **Option Button**

Use this selection to add option or radio buttons to a parent control. Use option buttons when the options are mutually exclusive for selection by the user.

Option buttons placed on a form are mutually exclusive for the entire form; i.e., only one may be selected by the user on a given form. Use the Button Group control when option buttons need to be mutually exclusive within a group (see below).

For properties, see Option Buttons.

 **List Box**

Use this selection to add a single column list box to a parent control that can be used for display or data value selection by the user. If more items are displayed than can be contained by the size of the list box on the, scroll bars will appear. If a list is to contain many values, it might be desirable to use a drop-down list box (see below) to conserve space on the form.

For properties, see List Boxes.

 **Drop-Down List Box**

Use this selection to add a drop-down list box to a parent control that can be used for display or data value selection by the user. The drop-down list box occupies only the amount of space on the form that it takes to display a single item in the list. Other items are made visible by clicking the drop-down arrow on the box.

For properties, see Drop-down List Boxes.

 **Multi-Column List Box**

Use this selection to add a multi-column list box to a parent control. Unlike the single column list box, the multi-column list box control supports header options that are displayed as part of the form.

For properties, see Multi-column List Boxes.

 **Bevel**

Use this selection to add a bevel to a parent control. The bevel has very few properties. It is primarily used to for appearance.

You can place controls within a bevel but those controls, unlike those on a panel or button group, may not be moved as a group.

For properties, see Bevel.

 **Button Group**

Use this selection to add an option button group to a parent control. When the button group is first placed on the form, no option buttons appear in the group. Use the Items property to add the option buttons. Next, use the button group sizing handles to size the group to contain the option items entered. The Item Index property can be used to specify which option in the group is initially set when the form is loaded (e.g., -1 means that no option is set; 0 means the first option is set; 1, the second, etc.).

For properties, see Button Groups.

 **Group Box**

Use this selection to add a group box to a parent control. Group boxes are generally used to logically group controls on the form (e.g., a group of controls that presents the user the Account Number, Balance, Amount Due, etc. might be placed in a group box captioned, Account Information).

You can place controls within a group box but those controls, unlike those on a panel or button group, may not be moved as a group.

For properties, see Group Boxes.

 **Panel**

Use this control to add a panel to a parent control. Controls can be placed on a panel and moved as a group when you select the panel and drag it to another location on the form. To place a control on a panel, select the panel on the form with the mouse. Next, click the Panel button on the DialogForm Designer toolbar and then click the mouse again over the panel.

To cut a control from another location (on the form or other panel) and paste it on a panel, select the control and enter Ctrl+X on the keyboard. Next, with the mouse, select the panel that is to receive the control and press Ctrl+V. Note: The control will be placed on the panel in the same relative location on which it was originally located; therefore, it may be necessary to increase the size of the panel in order to see the control.

For properties, see Panel.

 **Splitter**

Use this control to add a splitter to a parent control. A splitter, like those seen in popular Web browsers, allows the user to adjust the panels on a form to their own liking.

Splitter must be aligned and placed between other aligned controls. For example, if a splitter is to be placed vertically between two panels the left panel could be aligned left, followed by aligning the splitter left also. The right-hand panel then could be aligned to fill the remaining portion of the client area.

For properties, see Splitter.

 **Image**

Use this control to add an image to a parent control. Only pictures with .bmp, .wmf, .emf or .ico extensions are supported.

For properties, see Image.

 **Media Player**

Use this selection to add a media player control bar to a parent control. To use the Media Player control, use the LoadMMFile function in an action.

For properties, see Media Players.

 **Date/Time Label**

Use this selection to add a date/time stamp to a parent control. It is useful in a status bar (a bottom aligned panel).

For properties, see Date/Time Labels.

 **Menu Designer**

Use this selection to open the Dialog Form Menu Designer window to add or edit menus on the form.

Dialog Form Script Debugger

The Dialog Form Script debugger is very useful when developing Dialog Forms. The debugger can be used in both the Dialog Form Designer while developing a Dialog Form, or at run-time in the eXpress application in which the script is to be used.

When executing a Dialog Form in the debugger, the first thing that will happen is the Dialog Form Script Debugger window containing a copy of your event action script will be displayed. At this point, your Dialog Form will not be executing, but will allow you to set breakpoints before the script starts.

Breakpoints are placed in your event action script where you want to stop execution to examine values of variables and/or continue in single step (statement-by-statement) mode. To set breakpoint, simply move the edit cursor to any line containing a statement in your event action script and click the "Toggle Brk Pt" button. A red box will appear next to lines at which breakpoints are set. You may set as many breakpoints as you like.

When you click "Run," the Dialog Form will begin executing. If a breakpoint is encountered, execution will be suspended BEFORE that breakpoint statement is executed. At this point, you may examine the content of variables using the "Evaluate" button. You may continue executing the script by clicking

"Run" in which case the script will continue to run until another breakpoint is encountered or the script ends. Optionally, you may change to Single Step Mode. In Single Step mode, each time you click run, execution will pause after each statement is executed. Stepping allows you to watch the path your script takes while executing. To turn off Single Step mode, click "Single Step ON" (it becomes "OFF") then simply click "Run".

Script Debugger Tool Bar

Run	Start or resume script execution or execute one statement depending on the Single Step mode.
Single Step On/Off	Toggle the state of Single Step mode.
Toggle Brk Pt	Toggle a breakpoint at the current (cursor) statement.
Clear Brk Pts	Clear ALL breakpoints.

Evaluate Section

In this section, you may view the value of any variable or valid expression in the executing script. Evaluation can be done only when script execution is suspended at a breakpoint.

Property Editor

The Property Editor window is where the properties and event actions for individual controls placed on the dialog form are set. Correspondingly, the window contains two tabs: Properties and Event Actions. The Properties tab is where you name the control, as it will be referenced in dialog form event actions and initially set how the control will appear and behave to the user at runtime. The Event Action tab is used to assign actions when certain events take place with the control.

The dialog form can also be considered a control and, as such, may have properties assigned or changed by first selecting the form (left click on any empty space on the form). The dialog form also has event actions associated with it (see Form Properties).

Note: The dialog form is a "parent" control. Any form property (Font, Ctl3d, etc.) will be assigned to a control placed on the form if that control has its corresponding "parent" control set to True (see Parent Controls).

To change a property, first select the control with the mouse on the dialog form. With a control selected, the properties for the control will appear on the Properties tab. The left-hand column contains the property name; the right-hand column, the property value. Next, select a property with the mouse. A control (button, text box, etc.) will appear in the cell to the right of the property name that will allow you to change the property value.

Dialog Form Event Action Editor

This window provides the means to develop and edit dialog form actions. The editor contains multiple tabs that allow more than one action file to be edited at a time.

Following is a description of each Dialog Form Action Editor command. All the commands may be performed by making a menu selection. Toolbar buttons, shortcut keys and right mouse click actions are available for frequently used commands. A right mouse click anywhere in the test area will produce a pop-up menu of commands.

The menu items below show an image of the toolbar button and the shortcut key combination (in parentheses) where applicable.

File menu

The File menu contains commands to maintain action files and setup printing.

New (Ctrl+N)

Use this command to create a new action file (.ACT).

Open (Ctrl+O)

Use this command to open an existing action file.

Save (Ctrl+S)

Use this command to save the current action file.

Save As...

Use this command to save the current action file to another file name.

Close

Close the currently selected action file.

Close All

Close all open action files.

Print (Ctrl+P)

Use this command to print the entire action file.

Printer Setup...

Allow margins to be set and allow printers and printer fonts to be selected for printing.

Editor Properties

Use this command to edit the properties of the action editor: window font, highlight colors and tab stops.

Exit

Exit the Action Editor.

Edit menu

The Edit menu contains commands to manage selected text between the editor and the Windows clipboard.

Undo (Ctrl+Z)

Use this command to reverse the effects of the most recent change.

Redo (Ctrl+Shift+Z)

Use this command to reverse the effects of the most recent **Undo** command.

Cut (Ctrl+X)

Use this command to place the selected text on the clipboard and delete.

Copy (Ctrl+C)

Use this command to copy the selected text to the clipboard.

Paste (Ctrl+V)

Use this command to paste the contents of the clipboard to the current cursor position.

Delete (Ctrl+D)

Use this command to delete the selected text without copying to the clipboard.

Word Wrap (Ctrl+W)

Use this command as a toggle. By default, long lines may only viewed/edited by first bringing the excess text into view with the horizontal scroll bar or by using the cursor keys (arrows). When Word Wrap is set, long lines wrap to the next line and are viewable within the confines (width) of the window.

Search menu

The Search menu contains commands to locate and change text within the action file.

Find... (Ctrl+F)

Use this command to enable the **Find** dialog used to locate text strings.

Find Again (F3)

Use this command to find the next occurrence of the same string used on the previous find.

Replace... (Ctrl+R)

Use this command to enable the **Replace** dialog used to locate and replace text strings.

Go to Line (Ctrl+G)

Use this command to go to a specific line.

Bookmarks

The Bookmarks menu contains commands to mark lines and navigate within the action file.

Set Bookmark 1 through 5 (Shift+F1 through Shift+F5)

Use one of these commands to mark a line at the current text cursor position. A book marked line will appear with a gray background.

Go to Bookmark 1 through 5 (Ctrl+F1 through Ctrl+F5)

Use one of these commands to go to a line previously book marked by one of the five corresponding **Set Bookmark** commands.

Options menu

The Option menu contains commands to specify color, font and tab stop preferences.

Show Tool Bar

Use this command to toggle the display of the toolbar.

Syntax Highlight

Use this command to toggle the display of the action syntax. This command is affected by the settings of the **Editor Properties** command, above.

Show Class and Function Helper

Use this control to display a complete list of classes, class properties, types and constants. Also shown are all functions/procedures by category.

Note: If any item in a tree view is expanded all the way and selected, you can use a right mouse click to get the a "Copy Selected" pop up. "Copy Selected" copies the entire selected line to the clipboard.

Tools

The **Tools** menu contains commands to check action syntax.

Check Script (F4)

Use this command to check the syntax of the entire action file.

Help

The **Help** menu contains commands to display on-line help and information about the product.

This Window

Use this command to receive on-line help for this window.

About...

Use this command to display copyright and product version information.

Editor Properties

This dialog is used to change the properties or appearance of a script window.

Edit Window Font

The controls in this group affect the font typeface, size and intensity used to display the script.

Font Name

From this drop-down list box, choose from the list of non-proportional, fixed fonts installed on your PC.

Size

With this spin wheel, increase or decrease the font size.

Bold

Check this box to increase the font intensity.

Tab Size

With this spin wheel, increase or decrease the number characters between tab characters.

Highlight Colors

Use this group to assign colors to different parts of the script text.

Set Text Color

To change color, select the type of text (Normal text, Strings, etc.) and select from the Set Text Color drop-down list box to change the foreground.

Set Background Color

To change the background color, select from the Set Background Color drop-down list box.

OK

Click this button to accept the changes made and exit the dialog.

Cancel

Click this button to discard the changes made and exit the dialog.

Help

Click this button to receive on-line help for this dialog.

Controls on Dialog Forms

Form Properties

Form properties are set in the same manner as any control, from the Property Editor. Select form properties in one of two ways: 1) click the mouse on a blank portion of the form or 2) select "Form" from the drop-down list box at the top of the Property Editor dialog.

In addition to several event actions (Click, KeyPress, etc.) common to other contriols, there are four unique event actions associated only with a form: SetEventActions, FormShow, FormActivate and FormClose.

SetEventActions

The SetEventActions Sub is where the Form Designer (the program, not the person) will put all automatically generated Event Setup code.

Note: In general, you should not alter what is in SetEventActions.

FormShow Event Actions

FormShow event actions are fired just before the form is shown. Place any first-time code in this sub.

FormActivate Event Actions

FormActivate event actions take place when the form regains focus.

FormClose Event Actions

FormClose event actions are executed when the form is closed.

Form Properties

Forms have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
AutoCenter	Boolean	Y	Y
BackColor	TColor	Y	Y
BorderStyle	Integer	Y	Y
Caption	String	Y	Y
Ctl3d	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Left	Integer	Y	Y
MaximizedButton	Boolean	Y	Y
MinimizedButton	Boolean	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Width	Integer	Y	Y
WindowState	Integer	Y	Y

Text Label

Text labels can be used to create form titles, captions for edit boxes, captions for groups of controls, areas to receive messages from actions, and in general, a more Windows-like appearance to the form.

Event Actions associated with a text label are executed when the text label is clicked.

Text labels have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Alignment	Integer	Y	Y
Anchors	Integer	Y	Y
AutoSize	Boolean	Y	Y
BackColor	TColor	Y	Y
Border	Integer	Y	Y
Caption	String	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowAccelChar	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Top	Integer	Y	Y
Transparent	Boolean	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Edit Box

Event Actions associated with an edit box are executed when the user clicks in the edit box and/or when a change is detected in the box contents.

Edit Boxes have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Alignment	Integer	Y	Y
Anchors	Integer	Y	Y
AutoSize	Boolean	Y	Y
BackColor	TColor	Y	Y
CharCase	Integer	Y	Y
Ctl3d	Boolean	Y	Y
EditMask	String	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
MaxLength	String	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
PassWordChar	String	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Text	String	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Command Button

A command button is used to carry out a command or action when a user clicks the button. Any number of command buttons may be placed on a form or panel (see also, Panel).

Event Actions associated with a command button are executed when the button is clicked.

Command buttons have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Anchors	Integer	Y	Y
Bitmap	TBitmap	Y	
BitmapPosition	Integer	Y	Y
Cancel	Integer	Y	Y
Caption	String	Y	Y
Default	Integer	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
ModalResult	Integer	Y	Y
Name	String	Y	Y
NumBitMaps	Integer	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Speed Button

A speed button is a non-windowed control taking fewer system resources.

You can make a group of speed buttons act like radio buttons by setting the GroupIndex property to something other than 0.

The Down property specifies whether the button is down or up. The Down property is only in effect when the GroupIndex is not zero; otherwise, it will remain False. The AllowAllUp property affects this property also.

Speed buttons can also have a flat property that makes the outline show on mouse fly-over.

Event Actions associated with a speed button are executed when the button is clicked.

Speed buttons have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
AllowAllUp	Integer	Y	Y
Anchors	Integer	Y	Y
Bitmap	TBitmap	Y	
BitmapPosition	Integer	Y	Y
Caption	String	Y	Y
Down	Boolean	Y	Y
Enabled	Boolean	Y	Y
Flat	Integer	Y	Y
Font	TFont	Y	Y
GroupIndex	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
NumBitMaps	Integer	Y	Y

Property	Type	Design-time	Runtime
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Check Box

A check box is used to display a true/false, on/off or yes/no option that the user can set or clear by clicking. A "3" in a check box indicates that it is selected, set to on/true/yes. Any number of check boxes on a form can be checked at one time.

Event Actions associated with a check box are executed after the check box's state has changed (i.e., from unchecked to checked, or checked to unchecked). If an action changes the state of a check box, the check box's action will be triggered.

Check boxes have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Alignment	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
Caption	String	Y	Y
Checked	Boolean	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

See also, Edit Mask.

Option Button

An option button (frequently referred to as a Radio button) is used in conjunction with other option buttons to offer multiple choices, from which the user can select only one. Option buttons may be placed directly on the form, on a panel or in a button group. The buttons will be mutually exclusive upon the control on which they are placed.

Event Actions associated with an option button are executed when the option button's state has changed. If an action changes the state of an option button, the option button's action will be triggered.

Option buttons have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Alignment	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
Caption	String	Y	Y

Property	Type	Design-time	Runtime
Checked	Boolean	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

List Box

A list box is used to display a list of items from which a user may view or select (to select from the list requires that an action be associated with the list box). The list box size on the form is not dependent upon the number of values to be placed in the box. If a list box contains more values than can be accommodated by the size of the list box, a scroll bar will appear making all items in the list accessible. If a list is to contain many values, it might be desirable to use a drop-down list box to conserve space on the form.

Event Actions associated with a list box are executed when an item is clicked or double-clicked (two separate actions) from the list.

List boxes have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	Boolean	Y	Y
Items	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Sorted	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Drop-down List Box

Like the list box, the drop-down list box is used to display a list of items from which a user can select; however, the drop-down list box takes up less room on the form, and as such, is useful when the selection list contains many values. The drop-down list box only occupies one line on the form. In addition, the drop-down list box is limited to a single column.

Event Actions associated with a drop-down list box are executed when an item is selected (clicked) from the list.

Drop-down list boxes have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	Boolean	Y	Y
Items	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Sorted	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Multi-column List Box

A multi-column list box is similar to the standard list box in usage and behavior; however, in addition to supporting multiple columns, it allows optional column headings.

Event Actions associated with a multi-column list box are executed when an item is clicked or double-clicked (two separate actions) from the list.

Multi-column list boxes have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
ColumnHeaders	Integer	Y	Y
Columns	N/A	Y	
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
NumericSort	Boolean	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y

Property	Type	Design-time	Runtime
ShowHint	Boolean	Y	Y
SortColumn	Integer	Y	Y
SortDescending	Boolean	Y	Y
Sorted	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

See also, Multi-Column List Setup.

Memo

A memo control is a multi-line edit box. The user can type and edit large amounts of text in a Memo control. Access to a Memo from an Action Script is similar to handling items in a List Box.

Memo controls have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Run-time
Align	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
BorderStyle	Integer	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Lines	String	Y	Y
MaxLength	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ReadOnly	Integer	Y	Y
ScrollBars	Integer	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
WantsReturns	Integer	Y	Y
Width	Integer	Y	Y
WordWrap	Integer	Y	Y

Bevel

A bevel may be used to enhance the appearance of the form.

Actions may NOT be associated with a bevel.

Bevel have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Anchors	Integer	Y	Y
Name	String	Y	Y

Property	Type	Design-time	Runtime
Shape	Integer	Y	Y
Style	Integer	Y	Y
Visible	Boolean	Y	Y

Button Group

Button groups are used to group option buttons logically onto a single control. In other words, when the options in a group need to be mutually exclusive (only one option may be selected), the button group should be used. Note: Option buttons may be placed on a panel when tab control is required between buttons.

When the button group is first placed on the form, no option buttons appear in the group. Use the Items property to add the option buttons. Next, use the button group sizing handles to size the group to contain the option items entered. The Item Index property can be used to specify which option in the group is initially set when the form is loaded (e.g., -1 means that no option is set; 0 means the first option is set; 1, the second; etc.).

Event Actions associated with a button group are executed when the state of any option button in the group has changed. If an action changes the state of an option button, the button group's action will be triggered.

Button Groups may be populated at run-time just like list boxes.

Button groups have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
ButtonStyle	Integer	Y	Y
Caption	String	Y	Y
Columns	Integer	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
ItemIndex	Integer	Y	Y
Items	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Group Box

A group box may be used to logically group or frame a set of controls. The group box may not be used to establish option groups, and as such, has no physical functionality.

Event Actions may NOT be associated with group boxes.

Group boxes have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Anchors	Integer	Y	Y

Property	Type	Design-time	Runtime
BackColor	TColor	Y	Y
Caption	String	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Panel

A panel may be used to house one or more controls in a logical group. The controls behave on the panel as if they were placed on a form. In other words, if you move the panel the controls on the panel with move with the panel. In addition, if a "parent" property of a control is set to True, the control will take the property of the panel, the parent.

Event Actions associated with a panel are executed when the panel is clicked. The panel must be enabled and visible to click it. A panel can be used as a big button since its caption text can wrap from line to line. Panel text can also be aligned left, right or centered. The caption of regular buttons can contain only a single line of text centered in the client area.

Panel have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Alignment	Integer	Y	Y
Anchors	Integer	Y	Y
BackColor	TColor	Y	Y
BevelInner	Integer	Y	Y
BevelOuter	Integer	Y	Y
BevelWidth	Integer	Y	Y
BorderStyle	Integer	Y	Y
BorderWidth	Integer	Y	Y
Caption	String	Y	Y
Ctl3d	Boolean	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Boolean	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Splitter

A splitter is used to allow the end user to adjust the viewing area of controls like list boxes and panels at run-time.

Event Actions may NOT be associated with splitters.

Splitter have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
AutoSnap	Integer	Y	Y
BackColor	TColor	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
MinSize	Integer	Y	Y
Name	String	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Image

Image not only can be used to enhance the appearance of a form but also can be functional Windows controls that when selected perform a prescribed action.

Event Actions associated with an image are executed when the image is clicked.

Image have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Anchors	Integer	Y	Y
AutoSize	Boolean	Y	Y
Center	Integer	Y	Y
Enabled	Boolean	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentShowHint	Boolean	Y	Y
Picture	TPicture	Y	
ShowHint	Boolean	Y	Y
Stretch	Integer	Y	Y
Top	Integer	Y	Y
Transparent	Boolean	Y	Y
TransparentColor	Integer	Y	Y
TransparentMode	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Media Player

The media player control bar can be placed on a form to activate and play a movie. Normally, the form on which the control is placed loads the multi-media file when the form is first displayed. The control itself has no action associated with it; therefore, some other event, like the initial form action or command button, must be used to load the media file. For example:

```
Sub FormInitial()
' Action for FormInitial
        Rslt = LoadMMFile("Player", "C:\Data\TESTx\SPEEDIS.AVI")
End Sub
```

"Player" is the name of the media player control as set in the Name property.

Event Actions may NOT be associated with the media player control.

The media player control has the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Anchors	Integer	Y	Y
Enabled	Boolean	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
MonoChromeButtons	Integer	Y	Y
Name	String	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	String	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Date/Time Label

The data/time label is used to supply the current date and time (machine time) to the user on the form or control. For example, this control could be placed on a panel as a part if a status bar.

Event Actions may NOT be associated with a date/time label.

Date/Time labels have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Alignment	Integer	Y	Y
BackColor	TColor	Y	Y
BlinkColor	Integer	Y	Y
Blinking	Integer	Y	Y
BlinkIntervalOff	Integer	Y	Y
BlinkIntervalOn	Integer	Y	Y
Enabled	Boolean	Y	Y
FlatColor	Integer	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Format	String	Y	Y
FrameStyle	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Top	Integer	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

See also, Date Time Format Editor.

Browser

The browser control can be placed on a form to embed the browser on a form. The URL is set at run-time by an action using the "SetString" statement.

```
Sub FormInitial()
' Action for FormInitial
    SetString "Browser_1", "http://www.kmsys.com"
End Sub
```

"Browser_1" is the name of the media player control as set in the Name property.

Event Actions may NOT be associated with the browser control.

The browser control has the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Anchors	Integer	Y	Y
Enabled	Boolean	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentShowHint	Boolean	Y	Y
ShowHint	Boolean	Y	Y
TabOrder	Integer	Y	Y
TabStop	Boolean	Y	Y
Top	Integer	Y	Y
URL	String	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

URL Link

URL links can be used to place a URL link on a form. A URL link is similar to a Text Label in that its caption appears on the form, but it differs in that when clicked, it opens a separate browser window to the URL specified in the URL property. Furthermore, a URL link differs from a Browser control in that no browser is embedded on the form.

Event Actions associated with a URL link are executed when the URL link is clicked.

Text labels have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Align	Integer	Y	Y
Alignment	Integer	Y	Y
Anchors	Integer	Y	Y
AutoSize	Boolean	Y	Y
BackColor	TColor	Y	Y
Caption	String	Y	Y
Enabled	Boolean	Y	Y
Font	TFont	Y	Y
ForeColor	TColor	Y	Y
Height	Integer	Y	Y

Property	Type	Design-time	Runtime
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentColor	Integer	Y	Y
ParentFont	Boolean	Y	Y
ParentShowHint	Boolean	Y	Y
ShowAccelChar	Boolean	Y	Y
ShowHint	Boolean	Y	Y
Top	Integer	Y	Y
Transparent	Boolean	Y	Y
URL	String	Y	Y
Visible	Boolean	Y	Y
Width	Integer	Y	Y

Control Properties

Align Property

Use this property to align and automatically size the control on the parent control. Predefined constants are alNone (default), alTop, alBottom, alLeft, alRight and alClient.

alTop and alBottom will align the control at the top or bottom edge of the parent control, respectively. The control will take the height of the parent control.

alLeft and alBottom will align the control at the left or right edge of the parent control, respectively. The control will take the width of the parent control.

alClient will align at the width and height of the parent control.

Alignment Property

This property is used to align the text in a control. Predefined constants are alLeftJustify, alRightJustify and alCenter (not applicable for edit boxes).

AllowAllUp Property

The AllowAllUp property is used to force all speed buttons in a group to be up by default.

AllowAllUp is used in conjunction with GroupIndex. If you want a group of speed buttons to toggle like car radio buttons, you can set the GroupIndex of all of all buttons in the group to the same number. By default, one in the group will always be down. AllowAllUp changes the default behavior so that all buttons in such a group can be up.

Anchors Property

Use this property to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. The Anchors property can contain any combination of the following:

akLeft, akTop, akRight, akBottom.

When its parent is resized, the control holds its position relative to the edges to which it is anchored. By default, almost all controls are anchored [akLeft, akTop].

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors are enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

AutoCenter Property

Use this control to cause the form to be centered on the desktop at run-time.

AutoSize Property

This property is used to size the control according to the size of the text or image placed in the control. The default is False.

AutoSnap Property

The AutoSnap property is used to determine if the splitter will automatically snap to the edge of a parent control. The default is True.

BackColor Property

Use this control to set the background (non-text) color of a control. The default for most controls is CP-BtnFace. The default for edit boxes is CP-Window.

For forms, there is a rule regarding the form's Ctl3d and BackColor properties. If the BackColor is CP-BtnFace, when you toggle Ctl3d from True to False, the BackColor will toggle from CP-BtnFace to CP-Window. If the BackColor is something else, the current color is not change when Ctl3d is toggled.

See also Predefined Constants.

BevelInner Property

Use this property along with the BorderWidth property to add an inner bevel to a Panel control. The BorderWidth is the number of pixels from the outer edge of the panel that the inner bevel will begin. Predefined constants are bvNone (default), bvLowered and bvRaised. See also, BevelOuter, BevelWidth, BorderStyle and BorderWidth.

BevelOuter Property

Use this property to apply a lowered or raised bevel on the outer edge on a Panel control. Predefined constants are bvNone, bvLowered and bvRaised (default). See also, BevelInner, BevelWidth, BorderStyle and BorderWidth.

BevelWidth Property

The BevelWidth property is used to specify the width of any bevel on a Panel control. The default for this property is 1 pixel. See also, BevelInner, BevelOuter, BorderStyle and BorderWidth.

Bitmap Property

Use this property to select an optional bitmap to place on a button face.

BitmapPosition Property

Use the BitmapPosition property to specify the position of the bitmap on the button face. Predefined constants are blLeft (default), blRight, blTop and blBottom.

BlinkColor Property

Use this property to set the color for blinking text on the DateTimeLabel control when the Blinking property is set to True. The default color is CP-Highlight. This color is only visible on the control if the Blinking property is set to True. See also, Blinking, BlinkIntervalOff and BlinkIntervalOn.

Blinking Property

Use the Blinking property to make the text on the DateTimeLabel control blink. The default is False. See also, BlinkColor, BlinkIntervalOff and BlinkIntervalOn.

BlinkIntervalOff Property

This property is used to set the interval (in milliseconds) that the text in the DateTimeLabel control will NOT be displayed in BlinkColor property color. The default is 500 milliseconds. This value is only used if the Blinking property is set to True. See also, BlinkColor, Blinking and BlinkIntervalOn.

BlinkIntervalOn Property

Use this property to set the interval (in milliseconds) that the text in the DateTimeLabel control will be displayed in the BlinkColor property color. The default is 500 milliseconds. This value is only used if the Blinking property is set to True. See also, BlinkColor, Blinking and BlinkIntervalOff.

Border Property

This property may be used to place a border around a Text Label or Image control. The default is False. For Text Labels, this control is impacted by the setting of the Ctl3d property.

BorderStyle Property

Use this property to set the border style for Forms, Memo and Panel controls. Predefined constants are bsNone, bsSingle, bsSizeable (default) and bsDialog. The bsSizeable and bsDialog settings only apply to forms.

The bsSizeable value displays the form as a standard resizable dialog.

The bsDialog value displays the form as a non-resizable dialog with a standard border. In addition, with the bsDialog value, the system menu (upper-left) will not be shown, and the Minimize, Maximize and Close buttons (upper-right) will not be shown.

The bsSingle value causes a non-resizable dialog with a single-line border to be displayed.

The bsNone value displays a non-resizable dialog with no border.

For Panel, also see, BevelInner, BevelOuter, BevelWidth and BorderWidth.

BorderWidth Property

Use the BorderWidth property in conjunction with the BevelInner property to place a border on a Panel control. The BorderWidth is the number of pixels from the outer edge of the panel that the inner bevel will begin. The default is 1 pixel. See also, BevelInner, BevelOuter, BevelWidth and BorderStyle.

ButtonStyle Property

This property may be used to select the button style for the buttons in a Button Group control. Predefined constants are bsRadio (default) and bsPush.

Cancel Property

Use the Cancel property to specify whether a button's Action executes when the Escape key is pressed. If Cancel is True, the button's Action executes when the user presses Esc. Although a Form can have more than one Cancel button, the form calls the Action only for the first visible button in the tab order. The default value is False. To set the initial tab order, select Form Edits | Tab Order from the Edit menu on the Dialog Form Designer Toolbar or right-click anywhere on the form and select Tab Order. You may also order the tab order at runtime by using the TabOrder property.

Caption Property

This property is used to set text that will appear as a caption on a control; e.g., the caption in blue at the top of a dialog, the text in a text label, etc.

To add an accelerator key, add an ampersand (&) in front of any character in the caption. That character will appear underlined and becomes the accelerator key for that control. Accelerator key values should be unique amongst all accelerator key values on the form. When an accelerator key is pressed at runtime, the control's action is executed. See also, ShowAccelChar.

Note: On Text Labels, the accelerator character has no effect, other that how it appears in the text.

Note: Most control captions are a single line of characters; however, Text Labels allow for multiple lines in the caption. In a Text Label, use the Carriage Return key (chr$(13)) to begin the next line.

Center Property

Use this property to center the picture in an Image control. The default is False.

CharCase Property

This property may be used to specify the case of the characters to be typed into an Edit Box. Predefined constants are ecNormal (default, allowing both uppercase and lowercase), ecUpperCase and ecLowerCase.

Checked Property

Use this property to initially or programmatically check a checkbox or set an option button. The default is False.

ColumnHeaders Property

Use this property to establish column headers on a Multi-column List control. The default is False. When set to True, use the Columns property to create header text.

Columns Property

This property may be used to alter the number of columns (default is 5), change the row height (default is 17 pixels), specify headers and dividers. See also, the ColumnHeaders.

Ctl3d Property

Use the Ctl3d property to set disable on enable a three-dimensional look on a control. The default is True. See also, ParentCtl3d and the notes for the BackColor.

DataSource Property

This property may be used to alter the data source for an Edit Box or Text Label control. The DataSource property must be a valid field name in the screen associated with the current form.

Default Property

Use the Default property to give focus to a Command Button control when the form is first displayed. The default is False. If True and the user hits the Enter key, the button will be clicked.

Down Property

This property is used to initially at design time or programmatically at runtime press down a Speed Button control. The default is False. When used in conjunction with the GroupIndex property, and when multiple Speed Buttons have the same group index value, setting this property to True will raise another Speed Button in the same group, like car radio buttons.

See also Predefined Constants.

EditMask Property

Use this property to change the input edit mask for the Edit Box control. See Edit Mask dialog.

Enabled Property

This property is used to enable or disable a control. The default is True (enabled).

Flat Property

Use this property to give a speed button a flat instead of raised appearance. The default is False. If this property is set to True, the outline of the button will appear upon mouse flyover.

FlatColor Property

This property may be used to change the color of the frame on a Date/Time Label control when the FrameStyle is set to fsFlat. The default is CP-BtnShadow.

Font Property

Use the Font property to change the font on a control's caption or the font of a parent control (see Parent Controls). See also, ParentFont.

ForeColor Property

Use this property to set the foreground (text) color of a control. The default for most controls is CP-BtnFace. The default for edit boxes is CP-Window.

For forms, there is a rule regarding the form's Ctl3d and BackColor properties. If the BackColor is CP-BtnFace, when you toggle Ctl3d from True to False, the BackColor will toggle from CP-BtnFace to CP-Window. If the BackColor is something else, the current color is not change when Ctl3d is toggled.

See also Predefined Constants.

Format Property

This property is used to change to format of the date and time on the Date/Time Label control. See the Date Time Format Editor.

FrameStyle Property

Use this property to change the frame style on a Date/Time Label control. The available styles are fsNone, fsFlat, fsGrove, fsBump, fsLowered, fsButtonDown, fsRaised, fsButtonUp, fsStatus (default) and fsPopup. If the fsFlat value is selected, the color of the frame may be changed with the FlatColor property.

GroupIndex Property

This property is used to group Speed Buttons on a parent control. When used in conjunction with the Down property, and when multiple Speed Buttons have the same group index value, setting this property to True will raise another Speed Button in the same group, car radio buttons.

Height Property

Use this property to change the height of the control. It is used in conjunction with the Top property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the Dialog Form Designer Toolbar help page).

HelpContext Property

This property may be used to assign help for a control from a standard windows help file (.hlp).

Hint Property

Use this property to add a hint to a control. At runtime, when the user holds the mouse cursor over the control, the hint will appear briefly. See ShowHint and ParentShowHint.

ItemIndex Property

The ItemIndex property is used to specify which button in a Button Group is to be on initially at runtime. The default is -1 (no button initially set). 0 is the first button, 1 the second, etc. The Items property must be set prior to setting this property.

Items Property

Use this property to name the buttons in a Button Group. Each line entered is a separate button name. Use the Return key between lines.

Left Property

This property may be used to change the horizontal starting position of the control. It is used in conjunction with the Width property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the Dialog Form Designer Toolbar help page).

Lines Property

Use this property to enter or edit in a Memo control at design time.

MaximizedButton Property

This property is used to add or remove the standard Windows maximize button to a form. The default is True.

MaxLength Property

This property is used to change the maximum length (in characters) of an Edit Box or Memo control. The default is 0 (no maximum). For a Memo control, this represents the total number of characters in the control including carriage returns (chr$(13)). See also, Memo control.

MinimizedButton Property

Use this property to add or remove the standard Windows minimize button to a form. The default is True.

MinSize Property

The MisSize property is used to specify the minimum size of the panes (in pixels) on either side of the Splitter control. The default is 30.

Set MinSize to provide a minimum size the splitter must leave when resizing its neighboring control. For example, if the Align property is alLeft or alRight, the splitter cannot resize the regions to its left or right any smaller than MinSize pixels. If the Align property is alTop or alBottom, the splitter cannot resize the regions above or below it any smaller than MinSize pixels.

Note: Always set MinSize to a value less than half the client width of its parent. When MinSize is half the client width of the splitter's parent, the splitter cannot move because to do so would be to resize one of the panes less than MinSize pixels.

ModalResult Property

When the ModalResult property of a Button is set to something other than none, clicking the button sets the ModalResult of the Dialog and closes the Dialog. Basically, it automates the process of setting ModalResult.

MonoChromeButtons Property

Use this property to change the buttons on the Media Player control to appear in monochrome as opposed to color. The default is False.

Name Property

This property is used to assign a label to a control that can be referenced in an action script.

NumBitMaps Property

Use this property to specify the number of bitmaps that are to be used on a Command Button or Speed Button when the button is enabled or disabled. The default is 1. The maximum is 4. This property is used in conjunction with the Bitmap property.

Buttons can show different images depending on the state of the button: Up, Disabled, Clicked and Down. The Bitmap property can reference a bitmap that is divided into four images of equal size, side-by-side in a row. The first image will be shown when the button is up or has focus; the second if the button is disabled, the third when the button is pushed and the fourth if the button remains down.

If there is only one image in the bitmap, this image is used for all four states.

Note: The lower left pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap that matches that lower left pixel will be transparent.

NumericSort Property

Use this property in conjunction with the SetNumericProp subroutine when sorting multi-column list boxes.

The property is Boolean, specifying that the data in the SortColumn property is is to be compared numerically. The default is False (alpha compare).

This property can be changed in the Form Designer or, dynamically, at runtime.

See also, SortColumn and SortDescending properties.

ParentColor Property

If this property is set to True, the control will use the same color property of the parent control. The default is False. Also, see Parent Controls.

ParentCtl3d Property

This property is used to specify if a control is to take on the three-dimensional look (Ctl3d property) of the parent control. The default is True. Also, see Parent Controls.

ParentFont Property

If this property is set to True, the control will use the same Font property of the parent control. The default is True. Also, see Parent Controls.

ParentShowHint Property

Use this property to specify if the control is to use the ShowHint property of the parent control. The default is True. Also, see Parent Controls.

PassWordChar Property

This property is used to specify a password character for an Edit Box control. If a password character is specified, the user will only see password characters as they type into the edit box. The default is no password character in which case all characters typed are visible.

Picture Property

This property is used to place a bitmap into an Image control. The type of files that may be loaded are Windows or OS/2 bitmap (.bmp), icon (.ico), Windows metafile (.wmf) Windows enhanced metafile (.emf) or JPEG compliant files (.jpg and .jpeg).

ReadOnly Property

The ReadOnly property is used to determine whether the Memo control may have data entered at runtime or will be "read only." The default is False (read/write).

ScrollBars Property

Use this property to add or remove scrollbars to a Memo control. Predefined constants are ssNone (default), ssHorizontal, ssVertical and ssBoth.

Shape Property

This property determines the shape of a Bevel control. Predefined constants are bsBox (default), bsFrame, bsTopLine, bsBottomLine, bsLeftLine, bsRightLine and bsSpacer.

ShowAccelChar Property

Use this property to display or not display accelerator characters in a Text Label control. The default is True.

The accelerator character is an ampersand (&). If the ShowAccelChar property is set to true and an ampersand is entered to the left of a character, the character will appear underlined in the text label. For example, "S&ample" would appear as "Sample".

Note: On Text Labels, the accelerator character has no effect, other that how it appears in the text.

ShowHint Property

The ShowHint property along with the Hint property determines if a brief pop-up is to be displayed when the user moves the mouse cursor over the control. The default is True. See also, ParentShowHint Property.

SortColumn Property

Use this property in conjunction with the SetNumericProp subroutine when sorting multi-column list boxes.

The property is an Integer in the range of 0 to columns -1. The default is 0 (the first column).

This property can be changed in the Form Designer or, dynamically, at runtime.

See also, NumericSort and SortDescending properties.

SortDescending Property

Use this property in conjunction with the SetNumericProp subroutine when sorting multi-column list boxes.

The property is Boolean, specifying that the sort order is descending. The default is False (ascending sort).

This property can be changed in the Form Designer or, dynamically, at runtime.

See also, NumericSort and SortColumn properties.

Sorted Property

Use this property to determine if the items in a List Box, Drop-down List Box or Multi-column List Box are to be sorted. The default is False.

Strech Property

This property may be used to force a picture to fit the size if the Image control. The default is False.

Style Property

Use this property to specify the style of a Bevel control. Predefined constants are bsLowered (default) and bsRaised.

TabOrder Property

This property may be used to set the tab order of the controls when the user clicks the tab key. The tab order is like an index beginning with 1 and incremented by 1. It is used in conjunction with the TabStop property. An alternate and quick way to set tab order on multiple controls is to select Form Edits | Tab Order from the Edit menu on the Dialog Form Designer Toolbar or right-click anywhere on the form and select Tab Order.

TabStop Property

Use this property to specify whether a control is to be a tab stop when the user presses the tab key. The default is True.

Text Property

The Text property may be used to place text into an Edit Box control. The text is for documentation purposes only and will not be displayed at runtime.

Top Property

Use the Top property to change the vertical starting position of the control. It is used in conjunction with the Height property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the Dialog Form Designer Toolbar help page).

Transparent Property

This property is used to determine if the background of a Text Label control is see-through. The default is False.

TransparentColor Property

Use this control to set the transparent color of an image control. he default is Red.

TransparentMode Property

Use this control to set the transparent mode of an image control. The mode may be tmAuto (default) or tmFixed.

URL Property

This property is used to set the URL link to be accessed by the browser. When this control is clicked at runtime, it automatically opens the browser to the URL specified. No user code is needed.

Visible Property

Use this property to make a control visible or hidden. The default is True (visible).

WantsReturns Property

This property determines how the return character is to be handled in a Memo control. If True, the application accepts return characters into the text and does not pass the return character to any other control. If false, return characters are not accepted by the Memo control and may be passed to another control such as a default button. The default is True.

Width Property

Use this property to change the width of the control. It is used in conjunction with the Left property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the Dialog Form Designer Toolbar help page).

WindowState Property

This property is used to set the window state of the form. Predefined constants are wsNormal (default), wsMinimized and wsMaximized.

WinHelpFile Property

This property may be used to enter the name of an externally developed help file (.hlp). Help files are normally placed in the installation directory. Note: Do not enter the .hlp extension.

WordWrap Property

Use this property to specify if words will wrap to the next line when a line is full on a Memo control. The default is True. If True, lines automatically wrap based on the width of the control.

Property Dialogs

Parent Controls

Most controls have "Parent" properties (ParentCtl3D, ParentFont and/or ParentShowHint). If a parent property is set to True, then the control takes on the property of the parent control. Currently, there are only two types of parent controls other than the form itself: the group box and the panel control. For example, if a button on the form has the ParentFont property set to True, the button caption will use the same font assigned to the form. Note: Changing the Font property on the button will automatically set ParentFont property to False.

A panel or a group box can have controls placed on it so that whenever you move the panel during the design phase, all the controls on the panel move with it. Likewise, if an action disables the panel at runtime, all the controls on the panel are also disabled since the panel is parent to all the controls placed on the panel.

Edit Mask

This dialog is used to apply the EditMask property to an edit box.

Input Edit Mask

The Input Edit Mask is the mask that is used to limit the data that can be put into a masked edit box. A mask restricts the characters the user can enter to valid characters and formats. If the user attempts to enter a character that is not valid, the edit box does not accept the character. Validation is performed on a character-by-character basis.

If no edit mask is specified, the end-user is not restricted, except by maximum length if specified.

If a Custom Edit Mask is selected, the Input Edit Mask text box may be used to specify a mask other than the standard field edit masks supplied with the Dialog Form Designer. The Input Edit Mask is a case-sensitive text box used to specify the type of input that will be allowed, and the position in the field where each character will appear.

A mask consists of three fields with semicolons (;) separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are saved as part of the data. The third part of the mask is the character used to represent a blank in the mask.

Part 1:

The first part of the Edit Mask can contain any of the following characters:

Character	Meaning in Mask
!	If an exclamation (!) character appears in the mask, leading blanks do not appear in the data. If an exclamation character is not present, trailing blanks do not appear in the data.
>	If a greater than (>) character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a greater than character is encountered.
<	If a less than (<) character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a less than character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a back slash (\) character is a literal character. Use this character when you want to allow any of the mask special characters as a literal in the data.
L	The "L" character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The "l" character permits only an alphabetic character in this position, but does not require it.
A	The "A" character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, and 0-9.
a	The "a" character permits an alphanumeric character in this position, but does not require it.
C	The "C" character requires a character in this position.
c	The "c" character permits a character in this position, but does not require it.

0	The zero (0) character requires a numeric character only in this position.
9	The nine (9) character permits a numeric character in this position, but does not require it.
#	The pound (#) character permits a numeric character or a plus or minus sign in this position, but does not require it.
:	The colon (:) character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the International settings of the Control Panel utility on your computer system, that character is used instead of the colon.
/	The slash (/) character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the International settings of the Windows Control Panel utility on your computer system, that character is used instead of the slash.

Part 2:

In the second part of the edit mask, the "0" character means that the mask is not saved as part of the data. The "1" character means that the mask is saved as part of the data. For example, a telephone number could have parentheses around the area code as part of the mask. If the second part of the edit mask is "0", the parentheses do not become part of the data, making the size of the field slightly smaller.

Part 3:

In the third part of the Edit Mask, the underscore (_) character may be used to automatically insert underscores in the edit box for positions that are not yet filled. You may change this character to any desired fill character or a space.

Examples:

Edit Mask	Display	Internal
\(999\)999\-9999;0;	(770)635-6363	7706356363
\(999\)999\-9999;1;	(770)635-6363	(770)635-6363
999-999;0;_	123-4__	1234

Pre-defined Standard Edit Mask

This list box contains pre-defined edit masks from which you may select one. If you wish a customized edit mask, type directly into the Input Edit mask text box.

Character for Blanks

In this text box, enter the character that will appear in the edit box in place of a blank value.

Save Literal Characters

Check this box to save mask characters as part of the data. See, "Part 2," above.

Test Input

Use this text box to test type input.

Multi-Column List Setup

This dialog is used to specify the number of columns, row height, column headers, column alignment and vertical and horizontal dividers of a multi-column list box.

Number of Columns

Use this spin box to set the number of columns in the list.

Row height

Use this spin box to specify the height of each row in the list. The value specified is in pixels.

Column Header Options

The options in this group are used to set column headers if any.

Show Column Headers

Check this box if column headers are to be shown at the top of the list.

Use Preset Column Headers

Select this option if you set the column headers with the Column Size and Header Data control below.

Use 1st Data Line as Column Headers

Select this option if the first line of data is to be used as the column header. Note: The Show Column Headers option must be set.

Dividers

The options in this group determine if dividers should be shown between rows and columns.

Horizontal Dividers

Set this option to show dividers between rows.

Vertical Dividers

Set this option to show dividers between columns.

Column Size and Header Data

This control allows you to label column headers and size columns.

To label a column, click the button at the top of a column. A text cursor will appear in the row below the button. Type the desired header name. Note: The Show Column Headers and Use Preset Column Headers options must also be set.

To size a column, place the mouse pointer on a column divider. The mouse pointer will change to a splitter (left/right-sizing arrow). Hold the left mouse button down and drag the splitter to the left to reduce the size of the column or to the right to increase the size of the column.

Column n Display Alignment

To justify a column heading, select the column with the mouse and click the Left, Right or Center button.

Date Time Format Editor

This dialog is used to establish the format of the date/time stamp shown in the DateTimeLabel control.

Date Time Format

This table shows the characters you can use to create user-defined date/time formats:

Character	Meaning
c	Display the date as ddddd and display the time as tt, in that order.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display the day as a full name (Sunday-Saturday).
ddddd	Display the date as m/d/yy.
dddddd	Display the date as dddd, mmmm d, yyyy (e.g., Friday, October 27, 2000).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If mm immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
y	Display the day of the year as a number (1-366).
yy	Display the day of the year as a two-digit number (00-99)
yyyy	Display the day of the year as a four-digit number (0000-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
t	Display the time as h:n AM/PM.
tt	Display the time as h:n:s AM/PM.
AM/PM	Use the 12-hour clock and display an uppercase AM/PM.
am/pm	Use the 12-hour clock display a lowercase am/pm.
A/P	Use the 12-hour clock display an uppercase A/P.

a/p	Use the 12-hour clock display a lowercase a/p

The following are examples of user-defined date and time formats:

Format	Display
m/d/yy	2/26/65
d-mmmm-yy	26-February-65
d-mmmm	26-February
mmmm-yy	February-65
hh:mm AM/PM	06:45 PM
h:mm:ss a/p	6:45:15 p
h:mm:ss	18:45:15
m/d/yy h:mm	2/26/65 18:45

Predefined Formats

Select from this list if you want to use a predefined date/time format. See Date Time Format, above, for a description of the various predefined formats.

Additional Properties

Additional Properties and Methods

In addition to the properties and methods documented here and listed in the property editor window (Dialog Form Designer), there may be several more useful properties and methods that are not currently documented. The following table may be used to find a full list properties and methods available to each control at run-time. Locate the control in the Class and Function Helper by its class name.

Control	Class Name (Class and Function Helper)
Bevel	TBevel
Bowser	TDFBrowser
Button Group	TDFButtonGroup
Check Box	TCheckBox
Command Button	TBitBtn
Date/Time Label	TDFDateTimeLabel
Dialog Form	TDFForm
Drop Down List	TComboBox
Edit Box	TDFEdit
Group Box	TGroupBox
Image	TImage
List Box	TListBox
Media Player	TMediaPlayer
Memo	TMemo
Menu Items	TMenu
Multi-Column List box	TMultiColumnListbox
Option Button	TRadioButton
Panel	TPanel
Speed Button	TSpeedButton
Splitter	TSplitter
Text Lable	TLabel
URL	TDFURL

Menu Designer

Dialog Form Menu Designer

This dialog is used to add user menu items to the form.

Once a menu has been updated with the Dialog Form Menu Designer, the action for the menu item may be updated by selecting the menu item from the dialog form displayed in the Dialog Form Designer. See Dialog Form Designer.

Menu Item Caption

This entry specifies the menu item's caption. Captions may include accelerator keys by typing an ampersand character (&) in front of the desired letter within the caption text.

Menu separators may be indicated by entering a hyphen (-) in the caption.

Blank menu items are not permitted.

Name

In this text box, enter the name of the menu or menu selection. This name can be referenced in an action.

Short Cut

From the drop-down list box, select a short cut key for the menu selection. The shot cut key is optional.

Insert Item

Insert a blank menu item before the currently selected menu item. All menu items below and including the current menu item will be pushed down one position.

Indent Level

Use the arrow controls in this group to raise or lower the level of the selected menu or menu item in the Menu Items list. A menu item without a preceding asterisk represents a menu on the form's menu bar. Clicking the right arrow lowers the item level as indicated by an added asterisk (*) preceding the item. Clicking the left arrow raises the item level. Using the left and right arrow keys on the keyboard will also raise or lower the item level.

An item of a higher level with items below it at the next lower level becomes a cascading menu. For example, clicking an item called "Updates" (denoted by a single asterisk preceding it) might reveal a cascading menu containing "Add", "Replace" and "Delete" (each preceded by double asterisks).

Move

Use the arrow controls in this group to position the menu selection up or down in the list. You may also move an item with the keyboard by holding down the shift key and pressing the up and down arrow keys.

Checked

Check this control to preset the menu item to its selected state. Note: Only menu items may be checked – menus may not.

Visible

Check this box to make the menu or menu selection initially visible.

Enabled

Check this box to enable the menu or menu item initially.

Preview

Click this button to open a small dialog that reveals the designed menu bar. Test the menu and menu selection by clicking the menu.

Menu Items

This list box contains the menu and menu items defined. To work with an existing item, select it with the mouse or up and down arrows. Use the Indent and Move controls to rearrange the menus and menu items.

BasicScript Elements

BasicScript Language Elements

Statements can be either language elements or Functions and Procedures/Subs.

Script Structure

A script written in BasicScript language has the following general structure:

The Main Script

#Language BasicScript

The **#Language** statement is required and MUST be the first line of every script. It specifies the language syntax and is used by the script compiler.

[**Imports** "*FileName*"[,"*FileName*"]...]

The **Imports** statement is optional but MUST always follow the **#Language** statement. The **Imports** statement is used to add script statements to the current script from other script files.

Also see, Using "Uses" and "Imports" directives.

[*Script global variable declarations*]

This section is optional and contains declarations of constants and/or variables that are globally visible to the entire script.

[*Script global Functions and Subs*]

This section is optional and contains Functions and/or Subs globally visible to the entire script.

Main Script Statements

This section must be present and contains the main body of script statements.

Function and Sub structure

Function *FunctionName* [(*Parameter*1 [, *Parameter2*...])] **As** *ResultType*

Sub *SubName* [(*Parameter1* [, *Parameter2*...])]

A Function declares and defines a procedure that can receive arguments and returns a value of a specified data type. A Sub (subroutine) also receives arguments, but does not return a value.

Parameter form:

[{ **ByVal** | **ByRef** }] *ParameterName* **As** *type* [= *DefaultValue*]

All Functions and Subs must begin with a declaration that defines optional parameters that may be passed to the Function or Sub and, in the case of Functions, the value returned.

Parameters are defined by **ParameterName** and **type** and are referenced within the Function/Sub body as local variables. Parameter **type** can be any valid variable type.

ByVal (default) indicates that only the value of the parameter is passed and any changes made to it are not made to the original variable.

ByRef indicates that the parameter is a reference to the original variable passed in, and changes are directly applied to, the original variable.

Optionally, parameters may be assigned *DefaultValues*. A default value will be used when the parameter is NOT supplied on the Function/Sub call.

[*Function/Sub local variable declarations*]

This section is optional and contains declarations of constants and/or variables locally visible within the Function or Script.

Function/Sub Script Statements

This section is required and contains the body of Function or Sub statements.

End {**Function** | **Sub**}

The **End** defines the end of a Function or Sub.

Function/Sub Examples:

```
Sub Test1(ByRef Str As string, ByVal InVal As Int = 99)
  Str = IntToStr(Inval)
End Sub
Function Test2(ByVal Str As string, ByVal InVal As Int = 99) As String
  Return(IntToStr(Inval))
```

```
End Function
...
  MyStr = ""
  Test1(MyStr) ' This will change MyStr to "99"
  MyStr = Test2(MyStr, 123) ' This will change MyStr to "123"
...
```

Operators

Relational Operators

>	Greater than
<	Less than
<=	Less than or equal
>=	Greater than or equal
<>	Not equal
=	Equal
IN	Included is set
IS	Is type

Arithmetic Operators

+	Add
-	Subtract
*	Multiply
/	Divide
\	Integer divide
MOD	Modulo
&	Concatenation
OR	Logical OR
XOR	Logical exclusive OR
AND	Logical AND

Comments

Comments can be added to scripts in two ways. First, the REM statement (for Remark) can be used to create a comment line. This is somewhat out dated but still works. The more common way is to use the single quote (') character. In BasicScript, everything following a single quote is treated as a comment.

Strings Delimiters

BasicScript uses the double quote mark (") to delimit string constants. For example:

```
MyStr = "This is a string constant"
```

Script Structure Example

The following example demonstrates most of the structure discussed above:

```
#Language BasicScript
Dim i As Int
Dim s As string
' This function returns the higher of the 2 numbers pass in.
Function Max (Number1 As Int, Number2 As Int) As Integer
  If Number1 > Number2 Then
    Return Number1
  Else
    Return Number2
  End If
End Function
' This sub displays a message box containing the result.
Sub DisplayResult
  MsgBox(IntToStr(Max(50, i)) + " Is the maximim value.")
End Sub
' **** This is where the script execution starts.
  Do
    s = InputBox("Enter an integer number.", "Number Test")
    If s <> "" Then
      If Not ValidInt(s) Then ' Validate what the user typed.
        MsgBox("'" + s + "' is invalid.  Try again.")
        Continue
```

```
        End If
        i = Val(s)
        DisplayResult
      End If
    Loop until s = ""
' **** This is where the script execution ends.
```

Basic Variables

BasicScript may have types (for example, dim i as Integer), or may have no types and even no variable declaration. In this case, a variable will have the Variant type.

BasicScript variables are declared using the **Dim** statement as follows:

Dim *VariableName* as *VariableType*

See Common Language Elements for information of Variable Names and Variable Types

BasicScript Language Statements

Assign Statement

There is no keyword in Basic for the **Assign —** it is implied by the **=** operator.

Example:

```
x = 123; ' Assign 123 to x
```

Break Statement

Break

Immediately exit (break) out of a loop statement (Do, For, While, etc.), unconditionally.

Continue Statement

Continue

Stop processing within a loop statement (Do, While, etc.) and go to the next iteration.

Delete Statement

Delete *designator*

Delete the designated object or variable.

Exit Statement

Exit

Exit the current Function, Sub or script.

Set Statement

[**Set**] *variable* **=** *expression*

Assign a value to a variable. The keyword "Set" is implied and not normally used.

Return Statement

Return [*expression*]

Exit the current function, optionally returning a value.

If Statement

If *expression* **Then**
  statements
[**ElseIf** *expression* **Then**
  statements]
[**Else**
  statements]
End If

Allow conditional statements to be executed in the code.

Select Statement

Select Case *expression*
  Case *value* **:** *statements*
  [**Case Else :** *statements*]
End Select

Execute one of the sets of statement(s) in the case, based on the test variable.

Do/Loop Statement

Do

statements

Loop {**Until** | **While**} *expression*

Repeat execution of one or more statements **While** or **Until** the *expression* is true.

For/Next Statement

For *variable* = *expression* **to** *expression* [**Step**] *expression*

statements

Next

Repeat the execution of a block of statements for a specified duration.

Try/Finally/Catch Statement

Try

statements

{**Finally** | **Catch**}

statements

End Try

Provide a way to handle some or all possible errors that may occur in a given block of statements, while still running code. Use **finally** to insure a statement is executed even if an error is encountered.

Examples:

This Try/Finally block ensures that the Ptr objects is deleted (Freed), even if an error occurred:

```
Ptr = New TXsPrinter(Self)
Try
  BeginDoc
  ...
  End Doc
Finally
  Delete Ptr
End Try
```

This Try/Catch block will catch an error, allow the script to process it, and continue:

```
Try
  ....
Catch
  MsgBox("An error was encountered while....
End Try
```

With Statement

With *designator*

statements

End With

Execute a series of statements making repeated reference to a single object or structure.

Example 1:

```
Ptr = New TXsPrinter(self)
With Ptr
  Font.Name = "Courier New"
  Font.Size = 9
End With
```

Example 2:

```
' Activate a new screen
With TermScreen
  If ScreenAvailable ("TIP1")  Then
    MsgBox ("TIP1 Available")
    If Not ScreenOpen("TIP1") Then
      MsgBox ("TIP1 NOT Open")
      ActivateScreen ("TIP1")
    End If
  End If
End With
```

PascalScript Elements

PascalScipt Language Elements

Statements can be either language elements or Functions and Procedures.

Script Structure

A script written in PascalScript language has the following general structure:

The Main Script

#Language PascalScript

The **#Language** statement is required and MUST be the first line of every script. It specifies the language syntax and is used by the script compiler.

[**Uses** "*FileName*"[,"*FileName*"]...]

The **Uses** statement is optional but MUST always follow the **#Language** statement. The Uses statement is used to add script statements to the current script from other script files.

Also see, Using "Uses" and "Imports" directives.

[**Var** *Script global variable declarations*]

This section is optional and contains declarations of constants and/or variables that are globally visible to the entire script.

[**Const** *Script global constant declarations*]

[*Script global Functions and Subs*]

This section is optional and contains Functions and/or Subs globally visible to the entire script.

Begin

Main Script Statements

End.

This section must be present and contains the main body of script statements. The main body of the script must be enclosed with the **Begin/End.** block keywords.

Function and Procedure structure

Function *FunctionName* [(Parameter1 [; Parameter2...])] **:** *ResultType*

Procedure *ProcedureName* [(*Parameter1* [; *Parameter2*...])]

A Function declares and defines a procedure that can receive arguments and returns a value of a specified data type. A Procedure also receives argurments, but does not return a value.

Parameter form:

[**Var**] *ParameterName* **:** type [= *DefaultValue*]

All Functions and Procedures must begin with a declaration that defines optional parameters that may be passed to the Function or Procedure and, in the case of Functions, the value returned.

Parameters are defined by **ParameterName** and **type** and are referenced within the Function/Procedure body as local variables. Parameter **type** can be any valid variable type.

Var indicates that the parameter is a reference to the original variable passed in, and changes are directly applied to, the original variable. By default, only the value of the parameter is passed, and any changes made to it are not made to the original variable.

Optionally, parameters may be assigned *DefaultValues*. A default value will be used when the parameter is NOT supplied on the Function/Procedure call.

[**Var** *Function/Procedure local variable declarations*]

This section is optional and contains declarations of constants and/or variables locally visible within the Function or Script.

Begin

Function/Procedure Script Statements

End;

This section is required and contains the body of the Function or Procedure statements.

Function/Procedure examples:

```
Procedure Test1(Var Str : string; InVal : Int = 99);
```

```
Begin
  Str := IntToStr(Inval);
End;
Function Test2(Str : string; InVal : Int = 99) : String;
Begin
  Return(IntToStr(Inval))
End;
...
  MyStr := '';
  Test1(MyStr); // This will change MyStr to "99"
  MyStr := Test2(MyStr, 123); // This will change MyStr to "123"
...
```

Statement Blocks

In Pascal syntax, multiple statements must be placed into blocks bounded by **Begin** and **End** statements. For example:

```
If x = 1 Then
  a := x; // This line is executed only when x = 1.
  b := 2; // This line is always executed.
If x = 1 Then
  Begin
    a := x; // Both lines
    b := 2; // are execute when x = 1.
  End;
```

Semicolons (;) are used to terminate statements in Pascal. In general, a semicolon must terminate all statements. Some exceptions are 1) a statement immediately preceding an **Else** statement and 2) the **Begin** statement block keyword. For example:

```
...
Begin
  x := (y * 9) + z;
  If x > 1 then
    MsgBox('X is greater than 1')  // NO semicolon allowed here.
  else
    Begin
      MsgBox('X is less than 1');
      x := 0; // Reset x
    End;
End;
```

Operators

Relational Operators

>	Greater than
<	Less than
<=	Less than or equal
>=	Greater than or equal
<>	Not equal
=	Equal
IN	Included in set
IS	Is type

Arithmetic Operators

+	Add
-	Subtract
*	Multiply
/	Divide
DIV	Integer divide
MOD	Modulo
OR	Logical OR
XOR	Logical exclusive OR
AND	Logical AND
SHL	Bitwise shift left
SHR	Bitwise shit right

Comments

Comments can be added to PascalScript using the double slash (//). In PascalScript, everything following a double slash is treated as a comment. Another way to designate comments is to enclose them between braces ({}). Comments enclosed between braces are limited to a single line.

Strings Delimiters

PascalScript uses the single quote mark (') to delimit string constants. For example:

```
MyStr := 'This is a string constant'
```

Script Structure Example

The following example demonstrates most of the structure discussed above:

```
#Language PascalScript
Var
  i : Integer;
  s : string;
{
This function returns the higher of the 2 numbers passed in.
}
Function Max (Number1 : Int; Number2 : Int) : Integer;
Begin
  If Number1 > Number2 Then
    Result := Number1
  Else
    Result := Number2;
End;
// This procedure displays a message box containing the result.
Procedure DisplayResult;
Begin
  MsgBox(IntToStr(Max(50, i)) + ' Is the maximim value.')
End;
Begin
// **** This is where the script execution starts.
  Repeat
    s := InputBox('Enter an integer number.', 'Number Test');
    If s <> '' Then
      Begin
        If Not ValidInt(s) Then // Validate what the user typed.
          Begin
            MsgBox('''' + s + ''' is invalid.  Try again.');
            Continue;
          End;
        i := Val(s);
        DisplayResult
      End;
  Until s = ''
// **** This is where the script execution ends.
End.
```

Pascal Variables

Unlike BasicScript and JScript, ALL variables used in PascalScript must first be declared.

PascalScript variables are declared under the **Var** statement as follows:

Var

VariableName : *VariableType*;

See Common Language Elements for information on VariableNames and VariableTypes.

PascalScript Language Statements

Var Statement

Var

The **Var** statement is used to indicate the beginning of one or more variable declarations.

Example:

```
Var
  x : Int;
  s : String
```

Const Statement

Const

The Const statements is used to indicate the beginning of one or more constant declarations.

Example:

```
Const
  CompName = 'My Company Name';
  Pi = 3.15159;
```

Assign Statement

There is no keyword in PascalScript for the **Assign —** it is implied by the **:=** operator.

Example:

```
x := 123; ' Assign 123 to x
```

Break Statement

Break

Immediately exit (break) out of a loop statement (Do, For, While, etc.) unconditionally.

Continue Statement

Continue

Stop processing within a loop statement (Do, While, etc.) and go to the next iteration.

Delete Statement

Delete *designator*

Delete the designated object or variable.

Exit Statement

Exit

Exit the current Function, Procedure or script.

If Statement

If *expression* **Then**
statements
[**Else**
statements]

Allow conditional statements to be executed in the code.

Case Statement

Case *expression* **of**
value **:** *statements*
[**Else**
statements]
End

Execute one of the sets of statement(s) in the case, based on the test variable.

Repeat Statement

Repeat
statements
Until *expression*

Example:

```
x := 1;
Repeat
  if MyStr[x] = ' ' then
    MyStr[x] := '_';
  Inc(x);
Until x > Length(MyStr);
```

While Statement

While *expression* **Do**
statements

Execute a series of statements as long as a condition is true.

Example:

```
x := 1;
```

```
While x <= Length(MyStr) Do
  Begin
    if MyStr[x] = ' ' then
      MyStr[x] := '_';
    Inc(x);
  End;
```

For Statement

For *variable* := *expression* { **To** | **DownTo** } *expression* **Do**

statements

Repeat the execution of a block of statements for a specified duration.

Example:

```
For x := 1 to Length(MyStr) Do
  Begin
    if MyStr[x] = ' ' Then
      MyStr[x] := '_';
  End;
```

Try/Finally/Except Statement

Try

statements

{**Except** | **Finally** }

statements

End

Provide a way to handle some or all possible errors that may occur in a given block of statement, while still running code. Use **Finally** to insure a statement is executed even if an error is encountered.

Examples:

This Try/Finally block ensures that the Ptr objects is Freed even if an error occurred.

```
Ptr = New TXsPrinter(Self)
Try
  BeginDoc;
  ...
  End Doc;
Finally
  Ptr.Free;
End;
```

This Try/Except block will catch an error and allow the script to process it and continue.

```
Try
  ....
Except
  MsgBox('An error was encountered while....
End;
```

With Statement

With *descriptor* **Do**

statements

Execute a series of statements making repeated reference to a single object or structure.

JScript Elements

JScript Language Elements

Statements can be either language elements or Functions.

Script Structure

A script written in JScript language has the following general structure.

The Main Script

#Language JScript

The **#Language** statement is required and MUST be the first line of every script. It specifies the language syntax and is used by the script compiler.

[Imports "*FileName*"[,"*FileName*"]...]

The **Imports** statement is optional but MUST always follow the **#Language** statement. The **Imports** statement is used to add script statements to the current script from other script files.

Also see, Using "Uses" and "Imports" directives.

[*Script global variable declarations*]

This section is optional and contains declarations of constants and/or variables that are globally visible to the entire script.

[*Script global Functions*]

This section is optional and contains Functions globally visible to the entire script.

Main Script Statements

This section must be present and contains the main body of script statements.

Function Structure

Function *FunctionName* ([*Parameter1* [, *Parameter2*...]])

A Function declares and defines a procedure that can receive arguments and optionally returns a value of a specified data type.

Parameter form:

ParameterName [= *DefaultValue*]

All Functions must begin with a declaration that defines optional parameters that may be passed to the Function.

Parameters are defined by **ParameterName** (type is always Variant) and are referenced within the Function body as local variables.

Optionally, parameters may be assigned DefaultValues. A default value will be used when the parameter is NOT supplied on the Function call.

If a JScript Function has no parameters, it must still have a set of parentheses. As in this example:

```
Function MySub()
```

If the parentheses are omitted, the compiler detects no error.

[*Function local variable declarations*]

This section is optional and contains declarations of constants and/or variables locally visible within the Function or Script.

Function Body Script Statements

This section is required and contains the body of Function statements.

Function Example:

```
Function Test1(Str, InVal As Int = 99)
  {
  Result = (IntToStr(Inval));
  }
...
  MyStr = "";
  MyStr = Test1(MyStr, 123) ' This will change MyStr to "123"
...
```

Statement Blocks

In JScript syntax, multiple statements must be placed into blocks bounded by braces ({}) or, as some call them, "curly brackets". For example:

```
If (x == 1) Then
  a = x; // This line is executed only when x = 1.
  b = 2; // This line is always executed.
If (x == 1) Then
  {
    a := x; // Both lines
    b := 2; // are execute when x = 1.
  }
```

Semicolons (;) are used to terminate statements in JScript.

Operators

Relational Operators

>	Greater than
<	Less than
<=	Less than or equal
>=	Greater than or equal
!=	Not equal
==	Equal
IN	In set
IS	Is type

Arithmetic Operators

+	Add
-	Subtract
*	Multiply
/	Divide
||	Logical OR
^	Logical exclusive OR
&&	Logical AND
%	Modulo
<<	Bitwise shift left
>>	Bitwise shift right

Comments

Comments can be added to JScript using the double slash (//). In PascalScript, everything following a double slash is treated as a comment.

Strings Delimiters

JScript uses the double quote mark (") to delimit string constants. For example:

```
MyStr = "This is a string constant";
```

Inserting JScript Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JScript, a string is started and stopped with either single or double quotes. In the example above, the string will be truncated to:

```
We are the so-called
```

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JScript will now output the proper text string:

```
We are the so-called "Vikings" from the north.
```

Here is another example:

```
document.write ("You \& I are singing!");
```

The example above will produce the following output:

```
You & I are singing!
```

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	Outputs
\'	single quote
\"	double quote
\&	ampersand
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

Script Structure Example

The following example demonstrates most of the structure discussed above:

```
#Language JScript
var i, s;
// This Function returns the higher of the 2 numbers pass In.
Function Max (Number1, Number2)
  {
  If (Number1 > Number2)
    Result = Number1
  Else
    Result = Number2;
  }
// This Function displays a message box containing the result.
Function DisplayResult()
  {
  MsgBox(IntToStr(Max(50, i)) + " Is the maximim value.");
  }
// **** This Is where the script execution starts.
  s = "X";
  Do
    {
    s = InputBox("Enter an integer number.", "Number Test")
    If (s != "")
      {
      If (!ValidInt(s)) // Validate what the user typed.
        {
        MsgBox("'" + s + "' is invalid.  Try again.");
        Continue;
        }
      i = Val(s);
      DisplayResult;
      }
    }
  While (s != "");
// **** This is where the script execution ends.
```

JScript Variables

JScript variables are declared using the **Var** statement as follows:

Var *VariableName* [, *VariableName*....]

JScript variables are all of the Variant type, thus no variable type is specified.

See Common Language Elements for information on Variable Names.

JScript Language Statements

Assign Statement

There is no keyword in JScript for the **Assign —** it is implied by the **=** operator.

Example:

```
x = 123; ' Assign 123 to x
```

Break Statement

Break

Immediately exit (break) out of a loop statement (Do, For, While, etc.), unconditionally.

Continue Statement

Continue

Stop processing within a loop statement (Do, While, etc.) and go to the next iteration.

Delete Statement

Delete *designator*

Delete the designated object or variable.

Return Statement

Return [*expression*]

Exit the current function optionally returning a value.

If Statement

If (*expression*)
  statements
[**Else**
  statements];

Allow conditional statements to be executed in the code.

Switch/Case Statement

Switch (*expression*)
 {
 Case *Value* **:** *statements*
 [**Case**....]
 }
[**Default :** *statements*]

Execute one of the sets of statement(s) in the case, based on the test variable.

Example:

```
...
Switch (x)
  {
  Case 1 : Tmp = "One";
  Case 2 : Tmp = "Two";
  Case 3 : {
    Tmp = "Three";
    Tmp = Tmp + IntToStr(x);
    }
Default :
  Tmp = "Default";
  }
...
```

Do Statement

Do
  statements
While (*expression*)

Repeat execution of one or more statements **While** the *expression* is true.

While Statement

While (*expression*)
  statements

Execute a series of statements as long as a condition is true.

For Statement

For (*InitialExpression*) ; (*ConditionalExpression*) ; (*LoopExpression*)
  statements;

Repeat the execution of a block of statements for a specified duration.

Example:

```
MyStr = "A B C D E F";
c = 0;
For (x = 1; x < 10; x++)
```

```
    {
    If (MyStr[x] == " ")
      Inc(c);
    }
  MsgBox("MyStr containts " + IntToStr(c) + " spaces.");
```

Try/Finally/Except Statement

Try

statements

{ **Finally** | **Except** }

statements

With Statement

With *Descriptor*

statements

Execute a series of statements making repeated reference to a single object or structure.

Common Elements

Common Language Elements

Variables

Internally, eXpress Script operates with the Variant type and is based on it. Nevertheless, you can use the following predetermined types in your scripts. eXpress Script variables may have declared types as described here, or may have no types and even no variable declaration (BasicScript and JScript, only). When a variable that has no declaration is used, it will have the Variant type.

Each supported, scripting language syntax has its own way of declaring variables. JScript does not use variable declarations. See individual language elements.

A variable name is a unique name assigned to the variable by the script's author. The name may only contain letters, numbers, $ or _.

Variable Types

Integer - A signed or unsigned whole number. Any of the following types may be used but will be treated the same as integer:

Byte

Word

Longint

Cardinal

TColor

Boolean - A boolean value.

Extended - A signed or unsigned fractional number. Any of the following types may be used but will be treated the same as Extended.

Real

Single

Double

TDate

TTime

TDateTime

String - A string of characters.

Variant - A variable of undetermined type. The type of a Variant is determined by usage. For example, if an Integer value is assigned to a Variant, its type will be Integer. If a string value is then assigned to the same Variant, it will become a string type.

Arrays - Arrays of variables are declared simply be adding a length specification to the declaration statement as follows:

ArrayName [[*LowerLimit*]..*UpperLimit*] **:** *VariableType*

LowerLimit is optional and specifies the lowest limit of the array. If *LowerLimit* is specified, *UpperLimit* specifies the highest index to the array. If *LowerLimit* is omitted, the lower limit is 0 and *UpperLimit* specifies the number of entries in the array.

In the following examples, an array of 5 integers is defined without and with a lower limit.

BasicScript:

```
Dim Nums1 [5] as Int
Dim Nums2[0..4] as Int
```

PascalScript:

```
Var
  Nums1 [5] : Int;
  Nums2 [0..4] : Int;
```

Explicit vs. Implicit Declatations

In both Basic and JScript you do not have to explicitly declare variables. Implicit references are convenient for streamlined code, but can lead to frustration when debugging. For example, if "TermScreen" were misspelled in a statement as follows:

```
Tmp = TernScreen.GetScreenText(2, 23, 45)
```

No compile error would occur, because the compile assumes that at some point during execution the variable "TernScreen" (note spelling) will be setup. Unfortunately, the resulting runtime error is interpreted as an "I/O error 105" — not exactly, what you would expect. If the

same error is made in a Pascal Script, the compiler because of Pascal's strict declaration requirements finds it immediately.

To force variables to be explicitally defined in BasicScript and JScript scripts, use the "Explicit" directive.

BasicScript example:

```
#Language BasicScript
Explicit
```

JScript example:

```
#Language JScript
Explicit
```

The "Explicit" line must start in position 1 and be placed anywhere in the script after the "#Language" line. "Explicit" has no effect on PascalScript since variables must be explicitally declared by definition.

Array index referencing

In all eXpress Script languages, indexes are specified in backets ([]).

BasicScript or JScript:

```
x = MyArray[y];
```

PascalScript:

```
x := MyArray[y];
```

Variable Scope

Variable Scope refers to how a variable may be used within a script. A script global variable is declared as part of the main body of a script — NOT within a Function or Sub/Procedure. Script global variables can be references anywhere within the script, including from within a Function or Sub/Procedure. Local variables are declared within a Function or Sub/Procedure. Local variables can only be referenced within the function or Sub/Procedure in which they are declared. If a local variable is given a name that has already been given to a global variable, references to it within the Function or Sub/Procedure will use the local declaration. Any references to the same variable name in the main body of the script will use the global variable.

Using "Uses" and "Imports" directives

Large scripts can be split into modules, and using the "Uses" directive in Pascal ("Imports" in BasicScript and JScript), be referenced from a main script. For example:

File unit1.pas:

```
uses 'unit2.pas';
begin
  Unit2Proc('Hello!');
end.
```

File unit2.pas:

```
procedure Unit2Proc(s: String);
begin
  ShowMessage(s);
end;

begin
  ShowMessage('initialization of unit2...');
end.
```

As you can see, you should write module name with file extension in quotes. The code placed in begin...end of the included module will be executed when you run the script.

In this example, you cannot use unit1 from within unit2. This will cause circular reference and infinity loop when compiling such script.

Using #language directive, you can write multi-language scripts. For example, one module may be written in PascalScript, another one - using JScript:

File unit1.pas:

```
uses 'unit2.pas';
begin
  Unit2Proc('Hello from PascalScript!');
end.
```

File unit2.pas:

```
#language JScript
function Unit2Proc(s)
```

```
{
  ShowMessage(s);
}

{
  ShowMessage("unit2 initialization, JScript");
}
```

Built-In Functions and Procedures/Subs:

The following built-in functions and procedures/subs are listed by type/category:

Conversion

Function DateTimeToStr(e: Extended): String
Function DateToStr(e: Extended): String
Function FloatToStr(e: Extended): String
Function HexToInt(HexVal : String) : Integer
Function IntToHex(i: Integer, Digits : Integer = 4) : String
Function IntToStr(i: Integer): String
Function Str(n : Variant) : Variant
Function StrToDate(s: String): Extended
Function StrToDateTime(s: String): Extended
Function StrToFloat(s: String): Extended
Function StrToInt(s: String): Integer
Function StrToTime(s: String): Extended
Function TimeToStr(e: Extended): String
Function Val(v : Variant) : Variant
Function VarToStr(v: Variant): String
Function VarTypeToStr(VarType : Variant) : String

Formatting

Function Format(Fmt: String; Args: array): String
Function FormatDateTime(Fmt: String; DateTime: TDateTime): String
Function FormatFloat(Fmt: String; Value: Extended): String
Function FormatMaskText(EditMask: string; Value: string): string

Date and Time

Function Date: TDateTime
Function DayOfWeek(aDate: DateTime): Integer
Function DaysInMonth(nYear, nMonth: Integer): Integer
Function EncodeDate(Year, Month, Day: Integer): TDateTime
Function EncodeTime(Hour, Min, Sec, MSec: Integer): TDateTime
Procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Integer)
Procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Integer)
Function IsLeapYear(Year: Integer): Boolean
Function Now: TDateTime
Function Time: TDateTime

String Handling

Function Asc(ch: Char): Integer
Function Chr(i: Integer): Char
Function CompareText(s1, s2: String): Integer
Function Copy(s: String; from, count: Integer): String
Procedure DeleteStr(var CurrStr: String; FromPos, count: Integer)
Procedure Insert(NewStr: String; var CurrStr: String; pos: Integer)
Function InStr(StartChar: integer = 1, SubStr : String; StrVal : String) : integer

Function LCase(s: String) : String
Function Left(StrVal : String, Count : Integer) : String
Function Len(s: String) : integer
Function Length(s: String): Integer
Function Lowercase(s: String): String
Function LTrim(s: String) : String
Function MakeString(Length : Integer, FillChar : Char = #32) : String
Function Mid(s: String, StartPos : Integer; Count : Integer) : String
Function NameCase(s: String): String
Function Ord(ch: Char): Integer
Function Pos(substr, s: String): Integer
Function ReplaceStrings(s: String, StrToReplace: String, ReplaceWith: String) : String
Function Right (s: String, Count : Integer) : String
Function RTrim(s: String) : String
Procedure SetLength(var S: String; L: Integer)
Function Space(Length : Integer) : String
Function Trim(s: String): String
Function UCase(s: String) : String
Function Uppercase(s: String): String

Mathematical

Function Abs(e: Extended): Extended
Function ArcTan(X: Extended): Extended
Function Cos(e: Extended): Extended
Function Exp(X: Extended): Extended
Function Frac(X: Extended): Extended
Function Int(e: Extended): Integer
Function Ln(X: Extended): Extended
Function Pi: Extended
Function Round(e: Extended): Integer
Function Sin(e: Extended): Extended
Function Sqrt(e: Extended): Extended
Function Tan(X: Extended): Extended
Function Trunc(e: Extended): Integer

File/Folder

Function CopyFile(SourceFile : String, DestFile : String) : Boolean;
Function RenameFile(CurrentFileName : String, NewFileName : String) : Boolean;
Function DeleteFile(FileName : String) : Boolean;
Function ExtractFilePath(FileName : String) : String;
Function ExtractFileName(FileName : String) : String;
Function ExtractFileExt(FileName : String) : String;
Function ChangeFileExt(FileName : String, NewExt : String) : String;
Function FileExists(FileName : String) : Boolean
Function FolderExists(FolderName : String) : Boolean;
Function CreateFolder(FolderName : String) : Boolean;
Function RemoveFolder(FolderName : String) : Boolean;

Misc.

Function AppActivate(WindowTitle : String) : boolean
Procedure Beep(BeepType : integer)
Function CalendarDialog(InitialDate : String, Control : TComponent, LargeSize : boolean = False) : String

Function CreateOleObject(ClassName: String): Variant

Procedure Dec(var i: Integer; decr: Integer = 1)

Function ExecuteProgram(ExeFile : String, Parameters : String = "", WaitForComp : Integer = 0) : boolean

Function GetFolderPath(CLSID : Integer) : String

Frocedure Inc(var i: Integer; incr: Integer = 1)

Function InputBox(Prompt : String = "", Title : String = "", DefautValue : String = "") : String

Function MsgBox(Msg : String, Icon : integer = 0, Title : String = "") : integer

Procedure RaiseException(Param: String)

Function Random: Extended

Procedure Randomize

procedure SendMail(Recipients: String, Subject: String, CcRecipients: String, BccRecipients: String, MessageText: String, Attachments: String, NoPrompt: Boolean)

Procedure SendKeys(Keys : String, WindowTitle : String = "", Delay : integer = 0)

Procedure Shell(ProgramFile : String, Parameters : String = "", StartInDir : String = "", Style : Integer = 1)

Procedure ShowMessage(Msg: Variant)

Function ValidDate(cDate: String): Boolean

Function ValidFloat(cFlt: String): Boolean

Function ValidInt(cInt: String): Boolean

Function VarArrayCreate(Bounds: Array; Typ: Integer): Variant

Procedure Wait(milliseconds : Integer)

eXpress Scripting Classes

eXpress Scripting Classes

A number of programming classes are provided to facilitate routine tasks such as printing, reading and writing files, interacting with terminal screens and interfacing with user dialogs. Most have defined properties and methods (Functions/Procedures/Subs) and are described below.

The eXpress Scripting Classes are:

- TTermScreen
- TXSPrint
- TXSLinePrinter
- TXSTextFile
- TDialogForm

For additional classes that perform common dialog tasks, see Common Dialog Classes.

For additional class that perform font settings and drawings, see Advanced Classes.

TTermScreen Class

The TTermScreen class encapsulates all the interaction between a script and the current terminal screen. The TTermScreen class object is automatically created and is global to the current script session and any dialog forms created by the current script session.

TTermScreen Properties

Name	Type	Usage	Description
BlockEndColumn	Integer	Read	The current marked block ending column.
BlockEndRow	Integer	Read	The current marked block ending row.
BlockMarked	Boolean	Read	Indicates whether or not a Cut/Paste block is marked.
BlockStartColumn	Integer	Read	The current marked block start column.
BlockStartRow	Integer	Read	The current marked block start row.
Column	Integer	Read	The current cursor column position.
Columns	Integer	Read	Number of columns in the current screen.
HoldMessages	Boolean	Read/Write	
KeyboardLocked	Boolean	Read	The current Keyboard lock state
MessageWaiting	Boolean	Read	The current Message Waiting state
ReceivedCount	Integer	Read/Write	
ReceiveMsg	Boolean	Read/Write	
Row	Integer	Read	The current cursor row position.
Rows	Integer	Read	Number of rows in the current screen.
ScreenName	String	Read	The Screen Name of the current screen.
ScriptResult	Boolean	Read/Write	This property is used in eXpress Component sign-on scripts to tell the component whether or not the sign-on was successful.

TTermScreen Methods

Function WaitForString (ExpectedString : String) : Integer

Cause the script to wait for the specified *string*.

Function WaitForSpecificString (Row : Integer, Col : Integer, Lng : Integer, ExpectedString : String) : Integer

Cause the script to wait for the specified *string* at the specified location on the screen.

Procedure EnterTextFromPrompt (Prompt : String)

Enter a prompt string at the current cursor position.

Function WaitString (Target : String, Col : Integer, Row : Integer, NotEqual : Integer = 0, TimeOut : Integer = 5) : Integer

Cause the script to wait for the specified String at the specified location with NotEqual and TimeOut values.

Procedure DoTerminalKey(Key : Integer)

Issue any of the supported T27 or UTS keystrokes.

Procedure Send (TextToSend : String)

Send key sequences to application windows.

Function GetScreenText (Col : integer, Row : Integer, Length : Integer) : String

Retrieve a text string from the specified positions within the logical screen.

Function GetScreenAttribute (Col : integer, Row : Integer) : Integer

Return Protected, Blink and Video Off attribute states and the specified column and row position.

Function GetScreenColor (Col : integer, Row : Integer) : Integer

Return a 2-digit hex number indicating the background and foreground color at the specified column and row position.

Function GetScreenLine (LineNumber : Integer) : String

Retrieve one logical line of the mapped terminal screen buffer.

Function GetLastMsg : String

Retrieves the last message received from the host or communication system.

Function GetScreenCount : Integer

Returns the number of screens currently configured.

Function GetScreenName(Index : Integer) : String

Returns the name of the screen at index. Index must be in the range 0 to ScreenCount – 1.

Example:

```
' Display a MsgBox containing the names
' of all configured screens indicating open screens.
c = TermScreen.GetScreenCount
s = ""
For x = 0 To c-1
  c = TermScreen.GetScreenName(x)
  If TermScreen.ScreenOpen(c) Then
    s = s + Chr(13) + c + "<OPEN>"
  Else
    s = s + Chr(13) + c
  End If  Next
MsgBox(s, 0, "Available Screens")
```

Function ScreenAvailable (*ScreenName* : String) : Integer

Determine if a screen is available.

Function ScreenOpen (ScreenName : String) : Boolean

Open a screen.

Procedure ActivateScreen (ScreenName : String)

Activate the specified screen, if it is available.

Procedure RefreshScreen

Repaint the screen in its entirety.

Procedure SetCursor (Col : Integer, Row : Integer)

Set the column and row position of the text cursor within the logical screen.

Procedure SetScreenText (Col : Integer, Row : Integer, length : Integer = 0, Value : String)

Set the string value of an area within the logical screen.

Procedure EnterText (Value : String)

Enter the specified string at the current cursor position on the screen.

Procedure MarkBlock (SCol : Integer, SRow : integer, ECol : Integer, ERow : Integer)

Mark a block of text on the screen to be subsequently copied to the Windows clipboard by the **CopyToClipboard** procedure.

Procedure CopyToClipboard

Copy the marked text to the Windows clipboard. This subroutine must be preceded by a **MarkBlock** procedure.

Procedure PasteFromClipboard

Paste the contents of the Windows clipboard to the current cursor position of the screen.

Function GetUserParam (Index : Integer) : String

Retrieve user information for a calling script.

Procedure SaveScreen (FileName : String)

Save an entire screen/form to a file.

Procedure LoadScreen (FileName : String)

Load an entire screen/form from a file.

Function HostIPAddress : String

Get the IP Address of the host.

Procedure PostAlert (Title : String, Msg : String, Level : Integer)

Post a message to the alert box.

Procedure SetSessionVar (VarName : String, VarValue : Variant)

Set the contents of a global session variable.

Function GetSessionVar (VarName : String) : Variant

Retrieve the current content of a global session variable.

Procedure SwitchToolBar (ToolBarName : String, ToolBarNumber : Integer = 1, ShowIt : Boolean = True)

Switch toolbar.

TXSLinePrinter Class

The TXSLinePrinter Class is a simplified encapsulation of printing functions where the printer is managed as a line printer instead of a full-page composition. The TXSLinePrinter class object must be created (see New or Create) before use. You should never create more than one instance of TXSLinePrinter at a time.

Properties

Name	Type	Usage	Description
BottomMargin	Integer	Read/Write	Indicates the sized of the bottom page margin in either inches or centimeters (see Metric property).
CharsPerLine	Integer	Read	Indicates how many (approximately) characters will fit on a line in the current font and margin settings.
Font	TFont	Read/Write	See TFont advanced Objects
LeftMargin	Integer	Read/Write	Indicates the sized of the left page margin in either inches or centimeters (see Metric property).
LinesPerPage	Integer	Read	Indicates how many lines will fit on the page in the current font and margin settings.
Metric	Boolean	Read/Write	When True indicates that margins are specified in Centimeters instead of inches.
Open	Boolean	Read	Indicates whether or not the printer is currently opened.
Orientation	Integer	Read/Write	Indicates the current printer page orientation. Use poPortrait or poLandscape.
RightMargin	Integer	Read/Write	Indicates the sized of the right page margin in either inches or centimeters (see Metric property).
TopMargin	Integer	Read/Write	Indicates the sized of the top page margin in either inches or centimeters (see Metric property).

WrapLines	Boolean	Read/Write	Indicates that lines too long to fit within the left and right margins are to be wrapped to the next line instead of truncated.

Methods

Procedure BeginDoc

Start a new printer document. The document remains open until the EndDoc method is called or the Current Script Session ends.

To start a new document you must call EndDoc or Abort first.

Procedure EndDoc

End the current printer document and sends it to the printer.

Procedure NewPage

Insert a page break in the current printer document.

Procedure PrintLine(Text : String)

Print a line of text using the *Text* parameter.

Procedure LineSpace(Count : integer)

Advance the line counter leaving one or more blank lines. The optional *Count* parameter indicates the number of lines to advance. If omitted, the *count* is defaulted to 1 line. *Count* is limited to 10 lines.

Procedure Abort;

Abort the current document.

TXSPrint Class

The TXSPrint class encapsulates all currently supported eXpress scripting printing operations. The TXSPrint class object must be created (see New or Create) before use. You should never create more than one instance of TXSPrint at a time.

Properties

Name	Type	Usage	Description
Canvas	TCanvas	Read/Write	See TCanvas advanced Ojects
Font	TFont	Read/Write	See TFont advanced Objects
Open	Boolean	Read	Indicates whether or not the printer is currently opened.
Orientation	Integer	Read/Write	Indicates the current printer page orientation. Use poPortrait or poLandscape.
PageHeight	Integer	Read	The page height in pixels.
PageWidth	Integer	Read	The page width in pixels.
PenWidth	Integer	Read/ Write	Use to set the width of the line drawing pen used in the LineTo and DrawRect methods.
PixelsPerInch	Integer	Read	The number of pixels per inch of the current printer page settup
SelectedPrinter	String	Read	The currently selected printer name
X	Integer	Read	Current page drawing x coordinate
Y	Integer	Read	Current page drawing y coordinate

Methods

Procedure BeginDoc

Start a new printer document. The document remains open until the EndDoc method is called or the current script session ends.

To start a new document you must call EndDoc or Abort first.

Procedure EndDoc

Ends the current printer document and send it to the printer.

Procedure NewPage

Insert a page break in the current printer document.

Function GetTextWidth (Text : String) : integer

Return the pixel width of the specified text using the current printer and font settings.

Function GetTextHeight (Text : String) : integer

Return the pixel height of the specified text using the current printer and font settings.

Procedure TextOut(x : integer; y : integer; Text : String)

Write the specified text to the printer page at the specified x and y pixel coordinates. The current drawing x and y positions are not changed.

Procedure MoveTo(x : integer; y : integer)

Change the current page drawing position to the specified x and y coordinates.

Procedure LineTo(x : integer; y : integer)

Draw a line on the current page from the current drawing x and y position to the specified x and y position. The line's width is determined by the PenWidth property.

Procedure DrawRect(Left : integer; Top : integer; Right : Integer; Bottom : Integer)

Draw a rectangle using the specified pixel coordinates.

Procedure Abort;

Abort the current document.

TXSTextFile Class

The TXSTextFile class provides an easy interface to read and write text files in eXpress Scripting. The TXSTextFile class object must be created (see New or Create) for each text file to be read or written.

TXSTextFile Properties

Name	Type	Usage	Description
EOF	Boolean	Read	Indicates to end-of-file state of the current file. Applied only to files opened for reading.
FileName	String	Read	The name of the current file.
IsOpen	Boolean	Read	Indicates to current open state of the file.
LastErrorCode	Integer	Read	Last system error code, if any, encountered by a file operation.
LastErrorMessage	String	Read	Last system error message, if any, encountered by a file operation. This is the text version of the LastErrorCode.

TXSTextFile Methods

function Open(FileName : String; FileMode : TXSTextFileMode) : Boolean

Open the specified file in the specified FileMode.

Available FileMode values are:

Value	Mode
fmRead	Open the file for reading
fmWrite	Open the file for writing
fmAppend	Open the file for writing and append new records to the end of the file when it already exists.

procedure Close

Close the currently open file.

function ReadLine(ErrorStatus : integer) : String

Return to next line from the currently opened file. The file's FileMode must be fmRead. If successful ErrorStatus will contain 0; otherwise, it will contain the system error code.

procedure WriteLine(ErrorStatus : integer; Line : String

Write the specified line to the currently open file. The file's FileMode must be either fmWrite or fmAppend. If successful, ErrorStatus will contain 0; otherwise, it will contain the system error code.

The following is an example of the TSXTextFile object used to copy one text file to another:

```
dim st as integer
dim s as string
dim cnt as integer

F1 = New TXSTextFile(Self)
F2 = new TXSTextFile(Self)

try
  If Not F1.open(termscreen.scriptfolder +"\Buttons.xs", fmRead)
Then
    MsgBox("File F1 open error: " + F1.LastErrorMessage)
    Exit
  End If

   F2.Open(TermScreen.ScriptFolder +"\AAAA.xx", fmWrite)

   while not F1.EOF
     s = F1.readline(st)
     if st <> 0 then
       msgbox("Error on input file: " + F1.LastErrorMessage,
mb_IconExclamation, "Input File Error")
       break
     else
       inc(cnt)
       F2.WriteLine(st, s)
       if st <> 0 then
         msgbox("Write error: " + F2.LastErrorMessage,
mb_IconExclamation, "Output File Error")
         break
       End If

     End If
   WEnd
Finally
   F1.free
   F2.free
End Try

MsgBox("Copied " + IntToStr(cnt) + " lines.", mb_IconInformation,
"Copy Done")
```

TDialogForm Class

The TDialogForm class provides a mechanism for an eXpress Script to create and display a custom dialog window to the end-user. The content and behavior of a Dialog Form window is determined be the eXpress Script developer using the Dialog Designer.

See also, ModalResult Clarification and More.

TDialogForm Methods

Function Create(Owner : TObject) : Variant

This method creates an instance of a TDialogForm class object. Owner must always be specified as "Self" to ensure that the instance will be properly disposed of if the Script fails to complete normally.

Example:

BasicScript/JScript:

```
MyDialog = New TDailogForm(Self)
```

PascalScript:

```
MyDialog := TDialogForm.Create(Self)
```

Procedure Free

This method disposes of the instance of the TDialogForm object. Once Freed, an object must not be accessed, unless it is created again.

Function LoadForm(FileName : String, Debug : Boolean = False) : Boolean

This method loads a Dialog Form from the specified file created using the Dialog Form designer. Set Debug to true to have the Dialog Form actions execute in debug mode. It the file does not exist or is invalid the result will be false.

Procedure SetVariable(VarName : String, VarValue : Variant)

This method allows the script to initialize the value of a variable defined in the Dialog Form's action script. The specified variable must be declared Global in the Dialog Form's action script.

Function GetVariable(VarName : String) : Variant

This method is used to retrieve the value of a global variable in a Dialog Form's action script. If the variable is not defined, the returned value will be an empty string.

Function ShowForm : integer

This method causes the Dialog Form to be shown, modally. Modal means that the current script will wait for the Dialog Form to be closed before continuing to execute. The result will be whatever is set by the Dialog Form.

Procedure ShowFormNonModal

This method shows a Dialog Form in a non-modal state, meaning the script does not stop and wait for a Dialog Form to close. To use a non-modal dialog, the script has to keep itself alive, using loops or something, until time to close the form.

Procedure ClearFrom

This method clears the current Dialog Form contents from the instance of the TDialogForm. This method can be called to reuse the current instance to a TDialogForm for a new DialogForm.

Procedure PrintForm

This method prints a copy of the current dialog form window.

The following are examples of using the TDialogObject in an eXpress Script:

BasicScript:

```
df = New TDialogForm(Self)
Try
df.LoadForm(ScriptFolder + "\\NEWDIALOGTEST.bfm", true)
rslt = df.ShowForm
If rslt = mrOk Then
MsgBox("You selected:" + df.Edit_1.Text, mb_iconinformation, "Result")
Else
  MsgBox("Cancelled", mb_iconinformation, "Result")
End If
Finally
  df.Free
End Try
```

PascalScript:

```
Var Df : variant;
Var Rslt : integer;
df = TDialogForm.Create(Self)

begin
Try
df.LoadForm(ScriptFolder + '\NEWDIALOGTEST.bfm', true);
rslt := df.ShowForm;
If rslt = mrOk Then
MsgBox('You selected:' + df.Edit_1.Text, mb_iconinformation, 'Result')
Else
  MsgBox('Cancelled', mb_iconinformation, 'Result');
Finally
  df.Free;
End Try
End.
```

JScript:

```
Var df, rskt

df = New TDialogForm(Self)
Try
df.LoadForm(ScriptFolder + "\NEWDIALOGTEST.bfm", true)
rslt = df.ShowForm
If rslt = mrOk Then
```

```
MsgBox("You selected:" + df.Edit_1.Text, mb_iconinformation, "Result")
Else
  MsgBox("Cancelled", mb_iconinformation, "Result")
End If
Finally
  df.Free
End Try
```

ModalResult Clarification and More

This topic covers several things to consider when working with Dialog Forms.

Using the ModalResult Property

ModalResult is a run-time only (not available in the designer) property of the TDialogForm.

To set the ModalResult, or any other property of the TDialogForm, programmatically you must use either "Self" or the DialogForm's internal name reference, which will always be "Dialog". For example:

```
Sub Btn_OKClick(Sender)
  ModalResult  = mrOK        ' This does nothing
  Self.ModalResult = mrOK    ' This sets the ModalResult.  The dialog will
                             ' close when the sub is exited.
  Dialog.ModalResult = mrOK  ' Same as above
End Sub
```

If you want to set ModalResult and close the form when not using the ModalResult property of a Button, you must set it, and then use the Hide method of the form. Consider the following:

```
#Language BasicScript
Sub Lst_AccountsDblClick()
' Action for Lst_AccountsDoubleClick
    Dialog.ModalResult = mrOK     ' Set the ModalResult
    Dialog.Hide                   ' Hide the dialog to return the modal result
End Sub

Sub FormShow(Sender)
    Lst_Accounts.ItemIndex = 0
End Sub

Sub SetEventActions
    Lst_Accounts.OnDblClick = AddressOf Lst_AccountsDblClick
    Dialog.OnShow = AddressOf FormShow
End Sub
```

There are OK and Cancel buttons on the form that have ModalResults, but no code for them. BThe design calls for a doubleClick on the Account list to do the same as the OK button. If Dialog.Close were used instead of Dialog.Hide, the modal result would not be returned.

See the ACCOUNTSELECTOR form (.BFM and .ACT) in the installed examples located in the scripts folder.

Setting the Color Property of DialogForm

To change the Color property of the DialogForm use:

```
Self.Color = clBlue
```

or

```
Dialog.Color = clBlue
```

Closing the Dialog

If you do not care about the ModalResult and just want to close the Dialog, call the TDialogForm's Close procedure like this:

```
Self.Close
```

Using "Sender" in Event Actions

The Sender parameter-pass to control event actions is usually the control that caused the event to fire (usually the control itself); however, a control's event actions can be associated with other controls or called form another function. For example:

```
Sub Button2Click(Sender)
  If Sender Is TPanel Then
    If Sender = Panel1 Then
      MsgBox("You clicked Panel1")
    End If
      ElseIf Sender Is TBitBtn Then
        MsgBox("You clicked " + TBitBtn (Sender).Caption)
```

```
      End If
End Sub
Sub Panel1Click(Sender)
  Button2Click(Panel1)
End Sub
Sub FormInitialize
' Setup event actions here
  Panel1.OnClick = AddressOf Panel1Click
  Button2.OnClick = AddressOf Button2Click
  Button3.OnClick = AddressOf Button2Click  ' Manually added to use the same
                                            ' event action as Button2
End Sub
```

When Panel1 is clicked, it's event action calls Button2Click (Button2's OnClick event action) passing itself as Sender. Button3's OnClick event is assigned manually to Button2's OnClick. As you can see in Button2OnClick, Sender can be used to determine how to process the even.

Type Casting

Also shown here is an exampled of Type Casting.

```
MsgBox("You clicked " + TBitBtn (Sender).Caption)
```

Sender is always declared as a general Object, not a specific Class. To access Sender's properties and methods, it must be cast to its specific Class Type. In this example, the "is" operator is used to determine the Class of sender.

```
ElseIf Sender Is TBitBtn then
```

"TButton" is a Class Type, NOT a control's name. Once Sender's Class Type determined, it can be Type Cast to the correct class. Attempting to access an Object using an incorrect Type Case will most likely result in run-time errors.

Common Dialog Classes

Common Dialog Classes

This topic includes the properties and methods associated with the following common dialog classes:

TOpenFileDialog
TOpenPictureDialog
TSaveFileDialog
TSavePicureDialog
TPrintSetupDialog
TPrintDialog
TFontDialog
TColorDialog

TOpenDialog Class

TOpenDialog displays a modal Windows dialog box for selecting and opening files. The dialog does not appear at runtime until it is activated by a call to the Execute method. When the user clicks Open, the dialog closes and the selected file or files are stored in the Files property.

Properties

Name	Type	Usage	Description
DefaultExe	String	Read/Write	The default file extension of one is not entered
FileName	String	Read/Write	Select file name
Files	Strings	Read	A list of selected files it multi-select is on
Filter	String	Read/Write	The file selection filter
InitialDir	String	Read/Write	The initial folder path
Options	Integer	Read/Write	See file options

File Dialog Options

Value	Meaning
ofReadOnly	Select the Open As Read Only check box by default when the dialog opens.
ofOverwritePrompt	Generate a warning message if the user tries to select a file name that is already in use, asking whether to overwrite the existing file (use with save dialogs).
ofHideReadOnly	Remove the Open As Read Only check box from the dialog.
ofNoChangeDir	After the user clicks OK, resets the current directory to whatever it was before the file-selection dialog opened.
ofShowHelp	Display a Help button in the dialog.
ofNoValidate	Disables checking for invalid characters in file names. Allow selection of file names with invalid characters.
ofAllowMultiSelect	Allow users to select more than one file in the dialog.
ofExtensionDifferent	This flag is turned on at runtime whenever the selected filename has an extension that differs from DefaultExt. If you use this flag in an application, remember to reset it.
ofPathMustExist	Generate an error message if the user tries to select a file name with a nonexistent directory path.
ofFileMustExist	Generate an error message if the user tries to select a nonexistent file (only applies to Open dialogs).
ofCreatePrompt	Generate a warning message if the user tries to select a nonexistent file, asking whether to create a new file with the specified name.
ofShareAware	Ignore sharing errors and allow files to be selected even when sharing violations occur.
ofNoReadOnlyReturn	Generate an error message if the user tries to select a read-only file.
ofNoTestFileCreate	Disable checking for network file protection and inaccessibility of disk

	drives. Applie only when the user tries to save a file in a create-no-modify shared network directory.
ofNoNetworkButton	Remove the Network button (which opens a Map Network Drive dialog) from the file-selection dialog. Apply only if the **ofOldStyleDialog** flag is on.
ofNoLongNames	Display 8.3-character file names only. This option is only valid if Options also includes **ofOldStyleDialog**.
ofOldStyleDialog	Create the older style of file-selection dialog.
ofNoDereferenceLinks	Disable dereferencing of Windows shortcuts. If the user selects a shortcut, assign the path and file name of the shortcut itself (the .LNK file) to *FileName*, rather than the file linked to the shortcut.
ofEnableIncludeNotify	(Windows 2000 and later) Send CDN_INCLUDEITEM notification messages to the dialog when the user opens a folder. A notification is sent for each item in the newly opened folder. You can use these messages to control which items appear in the folder's item list.
ofEnableSizing	(Windows 98 and later) Let the Explorer-style dialog be resized with the mouse or keyboard. By default, the dialog allows this resizing regardless of the value of this option. It is only required if you provide a hook procedure or custom template (old style dialogs never permit resizing).
ofDontAddToRecent	Prevent the file from being added to the list of recently opened files.
ofForceShowHidden	Ensure that hidden files are visible in the dialog.

Methods

Function Create(Owner : Object)

Function Execute : Boolean

TOpenDialog Example:

```
Dim Fd

Fd = New TOpenDialog(Self) 'Create an instance
Fd.InitialDir = "C:\\MyFolder"
Fd.DefaultExt = "txt"
Fd.Filter = "Text files (*.txt)|*.txt|All file types (*.*)|*.*"
Fd.Title = "Open File Dialog Example"
If Fd.Execute then
  MsgBox("You selected file: " + fd.FileName)
Else
  MsgBox("Cancelled")
End If
Delete Fd ' Release instance
```

TOpenPictureDialog Class

TOpenPictureDialog displays a modal Windows dialog box for selecting and opening graphics files. This component is just like TOpenDialog, except that it includes a rectangular preview region. If the selected image can be read, it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

TSaveDialog Class

TSaveDialog displays a modal Windows dialog box for selecting file names and saving files. The dialog does not appear at runtime until it is activated by a call to the Execute method. When the user clicks Save, the dialog closes and the selected file name is stored in the FileName property.

TSavePicureDialog Class

TSavePictureDialog displays a modal Windows dialog box for selecting file names and saving graphics files. This component is just like TSaveDialog, except that it includes a rectangular preview region. If the selected image can be read, it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region

TPrintSetupDialog Class

TPrinterSetupDialog displays a modal Windows dialog box for configuring printers. The contents of the dialog vary depending on the printer driver selected.

The dialog does not appear at runtime until it is activated by a call to the Execute method.

TPrintDialog Class

The TPrintDialog component displays a standard Windows dialog box for sending jobs to a printer. The dialog is modal and does not appear at runtime until it is activated by a call to the Execute method.

TPrintDialog example:

```
ps = New TPrintDialog(self)

If ps.execute Then
  ' Just show the page size in pixels of the selected printer and options
  MsgBox("Selected printer: " + Printer.SelectedPrinter)
  MsgBox("Page width: " + Str(Printer.Pagewidth) + "  Page height: " +
Str(Printer.PageHeight))
End If

Delete ps
```

TFontDialog Class

TFontDialog displays a modal Windows dialog box for selecting fonts. The dialog does not appear at runtime until it is activated by a call to the Execute method. When the user selects a font and clicks OK, the dialog closes and the selected font is stored in the Font property.

Properties

The TFontDialog has only one meaningful property, Font, which has the following properties:

Name	Type	Usage	Description
Name	String	Read/Write	The fonts name
Size	Integer	Read/Write	The font size in points
Style	Integer	Read/Write	fsNormal, fsbold, fsItalic, fsUnderline, fsStrikethrough
Color	Integer	Read/Write	The font color code

Methods

Function Execute : Boolean

Execute (show) the font dialog. The Font property will reflect selected font changes if the user clicks OK, otherwise the Font property is unchanged.

TColorDialog Class

The TColorDialog component displays a Windows dialog box for selecting colors. The dialog does not appear at runtime until it is activated by a call to the Execute method. When the user selects a color and clicks OK, the dialog closes and the selected color is stored in the Color property.

Properties

Name	Type	Usage	Description
Color	Integer	Read/Write	The color code
Options	Integer	Read/Write	See ColorDialog options.

ColorDialog Options

Option	Purpose
cdFullOpen	Display custom color options when the dialog opens.
cdPreventFullOpen	Disable the Define Custom Colors button in the dialog, so that the user cannot define new colors.
cdShowHelp	Add a Help button to the color dialog.
cdSolidColor	Direct Windows to use the nearest solid color to the color chosen.

cdAnyColor	Allow the user to select non-solid colors (which may have to be approximated by dithering).

Advanced Classes

Advanced Classes

The following classes provide the means to control advanced printing options such as print font selections and drawings on a page:

- TFont
- TCanvas
- TBrush
- TPen

TFont Class

The advanced TFont object is available in most controls and can be used to access properties not defined in the in other parts of this documentation.

Properties

Name	Type	Usage	Description
Color	Integer	Read/Write	Specifies the font's color
Height	Integer	Read/Write	Specifies the font's height in pixels
Name	String	Read/Write	Specified the name of the font
Orientation	Integer	Read/Wrote	Specifies the orientation of the font
Size	Integer	Read/Write	Specifies the font size in pixels
Style	Integer	Read/Write	Specifies the font style which can be any combination of the following: fsBold fsItalic fsUnderline fsStrikeThru

TCanvas Class

The TCanvas advanced object is available to the Printer class and provides many advanced ways to draw on the page.

Properties

Name	Type	Usage	Description
Brush	TBrush		See TBrush
CopyMode			
Font	TFont		See to TFont
Pen	TPen		See to TPen
Pixels	TColor	Read/Write	Indexed [x, y] to get or set the color of individual pixels on the canvas

Methods

Procedure Draw(x : integer, y : integer, Graphic : TGraphic)

Draws a graphic at the specified x, y coordinates.

Procedure Ellipse(x1 : integer; y1 : integer; x2 : integer; y2 : integer)

Draw an Ellipse within the bounds specified by x1, y1, x2 and y2 using the current pen and brush.

Procedure LineTo(x : integer; y : integer)

Draw a line using the current pen from the current x, y coordinates to the specified x, y coordinates.

Procedure MoveTo(x : integer; y : integer)

Move the current x, y coordinates to the specifies x, y coordinates.

Procedure Rectangle(x1 : integer; y1 : integer; x2 : integer; y2 : integer)

Draw a rectangle bounded by the specified x1, y1, x2, y2 coordinates using the current brush and pen.

Procedure RoundRect(x1 : integer; y1 : integer; x2 : integer; y2 : integer; x3 : integer; y3 : integer)

Draw a rounded rectangle bounded by the specified x1, y1, x2, y2 coordinates using the current brush and pen. The x3 and y3 specify the x and y radii of the corners.

Procedure StretchDraw(x1 : integer; y1 : integer; x2 : integer; y2 : integer, Graphic : T Graphic)

Draw a graphic within the specified x1, y1, x2, y2 coordinates. The graphic's dimensions will be stretched/shrunk to fit the specified rectangle.

Function TextHeight(Text : String) : integer

Return the pixel height of the specified text using the current printer and font settings.

Procedure TextOut(x : integer; y : integer; Text : String)

Write the specified Text string are the specified x, y coordinates;

Function TextWidth(Text : String)

Return the pixel width of the specified text using the current printer and font settings.

TBrush Class

The brush determines the color and pattern for filling graphical shapes and backgrounds.

Name	Type	Usage	Description
Color	Integer	Read/write	The brush color used in filling rectangles, ellipses, etc.
Style	Integer	Read/write	Specifies the brushes pattern fill style: bsBDiagonal bsClear bsCross bsDiagCross bsDiagonal bsHorizontal bsSolid (Default) bsVertical

TPen Class

Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

Name	Type	Usage	Description
Color	Integer		The pen color used in drawing lines, rectangles edges, etc.
Mode	Integer		Specifies the pen's drawing mode: pmBlack pmCopy (default) pmMask pmMaskNotPen pmMaskPenNot pmMerge pmMergeNotPen pmMergePenNot pmNot pmNotCopy pmNotMask pmNotMerge pmNotXor

		pmWhite pmXor
Style	Integer	Specifies the pen's drawing style: psClear psDash psDashDot psDashDotDot psDot psInsideFrame psSolid (default)
Width	TPen	Specifies the thickness of the pen in pixels

Pen Mode description:

Mode	Pixel color
pmBlack	Always black
pmWhite	Always white
pmNop	Unchanged
pmNot	Inverse of canvas background color
pmCopy	Pen color specified in Color property
pmNotCopy	Inverse of pen color
pmMergePenNot	Combination of pen color and inverse of canvas background
pmMaskPenNot	Combination of colors common to both pen and inverse of canvas background
pmMergeNotPen	Combination of canvas background color and inverse of pen color
pmMaskNotPen	Combination of colors common to both canvas background and inverse of pen
pmMerge	Combination of pen color and canvas background color
pmNotMerge	Inverse of pmMerge: combination of pen color and canvas background color
pmMask	Combination of colors common to both pen and canvas background
pmNotMask	Inverse of pmMask: combination of colors common to both pen and canvas background
pmXor	Combination of colors in either pen or canvas background, but not both
pmNotXor	Inverse of pmXor: combination of colors in either pen or canvas background, but not both

Functions and Procedures

Abort Procedure

Applies to: TXSPrint Class.and TXSLinePrinter Class.

Abort the current document.

BasicScript:

Sub Abort ()

PascalScript:

Procedure Abort;

JScript:

Function Abort()

Related Topics: EndDoc Procedure, NewPage Procedure, PrintLine Procedure, LineSpace Procedure, BeginDoc Procedure, EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

Abs Function

Return the absolute value of a numeric expression.

BasicScript:

Function Abs (By Val *e* as Extended) as Extended

PascalScript:

Function Abs (*e* : Extended) : Extended

JScript:

Function Abs (*e*)

The data type of the return value is the same as that of the *e* argument.

BasicScript Example:

```
Dim Msg, X, Y

X = InputBox("Enter a Number:","","")
Y = Abs(X)

Msg = "The number you entered is "  & X
Msg = Msg + ". The Absolute value of " & X & " is " & Y
MsgBox (Msg) ' Display Message.
```

ActivateScreen Procedure

Applies to: TTermScreen Class.

Activate the specified screen, if it is available.

BasicScript:

Sub ActivateScreen (By Val *ScreenName* as String)

PascalScript:

Procedure ActivateScreen (*ScreenName* : String)

JScript:

Function ActivateScreen (*ScreenName*)

The *ScreenName* parameter is a string expression that represents the configured screen name to be activated. The activated screen is still not available to the script. A script still open only has access to the screen from which it was started.

If an invalid *ScreenName* is entered, it is ignored.

Related Topics: ScreenAvailable, ScreenOpen

BasicScript Examples:

```
' Activate a new screen
If TermScreen.ScreenAvailable ("TIP1") Then
    ' Or If TermSceeen.ScreenAvalable = True
  MsgBox ("TIP1 Available")
  If not TermScreen.ScreenOpen("TIP1") Then
```

```
      MsgBox ("TIP1 NOT Open")
      TermScreen.ActivateScreen ("TIP1")
    End If
  End If
```

Or:

```
  ' Activate a new screen
  With TermScreen
    If ScreenAvailable ("TIP1")  Then
      MsgBox ("TIP1 Available")
      If Not ScreenOpen("TIP1") Then
        MsgBox ("TIP1 NOT Open")
        ActivateScreen ("TIP1")
      End If
    End If
  End With
```

AppActivate Function

Activate another Windows application.

BasicScript:

Function AppActivate (By Val *Application* as String) as String

PascalScript:

Function AppActivate (*Application* : String) : String

JScript:

Function AppActivate (*Application*)

The *Application* parameter is a string expression and is the name that appears in the title bar of the application window to be activated.

Related Topics: Shell, SendKeys

BasicScript Example:

```
  AppActivate ("Microsoft Word")
  SendKeys ("%F,%N,Enable")
  Msg = ("Click OK to close Word")
  MsgBox (Msg)
  AppActivate ("Microsoft Word")  ' Focus back to Word
  SendKeys ("%F,%C,N")            ' Close Word
```

ArcTan Function

Return the arctangent of a numeric expression.

BasicScript:

Function ArcTan (ByVal *X* as Extended) as Extended

PascalScript:

Function ArcTan (*X* : Extended) : Extended

JScript:

Function ArcTab (*X*)

The *X* argument can be any numeric expression. The result is expressed in radians.

Related Topics: Cos, Tan, Sin

BasicScript Example:

```
  Dim Msg                ' Declare variable
  Pii = 4 * ArcTan(1)    ' Calculate Pi.
  Msg = "Pi is equal to " & FloatToStr(Pii)
  MsgBox (Msg)           ' Display results.
```

Note: Normally, you do not need to calculate Pi since Pi is a built-in function. The calculation of Pi in the above example is used simply to demonstrate the use of ArcTan.

Asc Function

Return the ASCII value of a character (Ord).

BasicScript:

Function Asc (By Val *String* as String) as String

PascalScript:

Function Asc (*String* : String) : String

JScript:

Function Asc (*String*)

Related Topic: Ord Function

BasicScript Example:

```
Dim I, Msg                    ' Declare variables.
For I = Asc("A") To Asc("Z")  ' From A through Z.
    Msg = Msg & Chr(I)        ' Create a string.
Next
MsgBox (Msg)                  ' Display results.
```

Beep Procedure

Produce a sound alert.

BasicScript:

Sub Beep (ByVal *BeepType* as Integer)

PascalScript:

Procedure Beep (*BeepType* : Integer)

JScript:

Function Beep (*BeepType*)

BeepType is a numeric expression equal to 0 (default) or set to one of the following:

BeepType	Constant
16	MB_ICONSTOP
32	MB_ICONQUESTION
48	MB_ICONEXCLAMATION

The frequency and duration of the beep depends on hardware, which may vary among computers.

BasicScript Example:

```
L = 0
Do
  Answer = InputBox("Enter a value from 1 to 3.", "", "")
  If (Answer >= 1) and (Answer <= 3) Then
    L = 1                              ' Set to exit Do Loop
  Else
    Beep (MB_ICONQUESTION)             ' Beep if not in range
  End If
Loop While L = 0
MsgBox ("You entered a value in the proper range.")
```

BeginDoc Procedure

Applies to: TXSPrint Class and TXSLinePrinter Class.

Start a new printer document. The document remains open until the EndDoc method is called or the current script session ends.

To start a new document you must call EndDoc or Abort first

BasicScript:

Sub BeginDoc ()

PascalScript:

Procedure BeginDoc

JScript:

Function BeginDoc()

Related Topics: EndDoc Procedure, NewPage Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

CalendarDialog Function

Return a date by showing a calendar dialog.

BasicScript:

Function CalendarDialog (ByVal *InitialDate* as String, ByVal *Control* as TComponent, ByVal *LargeSize* as Boolean = False) as String

PascalScript:

Function CalendarDialog (*InitialDate* : String, *Control* : TComponent, *LargeSize* : Boolean = False) : String

JScript:

Function CalendarDialog (*InitialDate*, *Control*, *LargeSize* as Boolean = False)

The *InitialDate* parameter is any string variable containing the date on the calendar to select, initially. The date is entered as a string in the YYYYMMDD format. It must be exactly eight characters in length. If an empty string is used, the current date is selected.

The *Control* parameter is the name of any existing control on a Dialog Form. It is used to force the Calendar Dialog to display with its upper left-hand corner aligned just to the right and below the named control's upper left-hand corner. Example:

```
NewDate = CalendarDialog(OldDate, Button1)
```

The *Control* parameter is only valid if the Calendar Dialog is being used within a Dialog Form. If it is not used, it MUST be specified as Nil in which case the Calendar Dialog will be centered on the screen. For example:

```
NewDate = CalendarDialog(OldDate, nil)
```

Note: Nil has a special meaning. When allowed, it can be used in place of any Object referenced.

The *LargeSize* parameter is True or False. Set to True, a large calendar dialog will be displayed.

The dialog simply displays a calendar with which the user can select a date. Initially, the calendar displays a single month, but the dialog may be expanded to show up to an entire year. The date is returned as a string in the format "YYYYMMDD". Canceling returns what ever was supplies as an initial date.

ChangeFileExt Function

Change a file's extension. The period (.) is consdered part of the extension.

BasicScript:

Function ChangeFileExt (ByVal *FileName* as String, ByVal *NewExt* as String) as String

PascalScript:

Function ChangeFileExt (*FileName* : String, *NewExt* : String) : String

JScript:

Function ChangeFileExt (*FileName*, *NewExt*)

This function will return the file name with the changed extension. It does NOT rename the actual file.

Chr Function

Returns the character represented by a specified integer value.

BasicScript:

Function Chr (ByVal *integer* as Integer) as Char

PascalScript:

Function Chr (*integer* : Integer) : Char

JScript:

Function Chr (*integer*)

Chr returns a String

BasicScript Example:

```
Dim X, Y, Msg, NL
NL = Chr(10)
For X = 1 to 2
  For Y = Asc("A") To Asc("Z")
    Msg = Msg & Chr(Y)
  Next
  Msg = Msg & NL
```

```
Next
MsgBox (Msg)
```

ClearForm Procedure

Applies to: TDialogForm Class.

This method clears the current Dialog Form contents from the instance of the TDialogForm. This method can be called to reuse the current instance to a TDialogForm for a new DialogForm.

BasicScript:

Sub ClearForm ()

PascalScript:

Procedure ClearForm

JScript:

Function ClearForm()

Related Topics: Free Procedure, LoadForm Function, SetVariable Procedure, GetVariable Function, ShowForm Function, Create Function, PrintForm Procedure

Close Procedure

Applies to: TXSTextFile Class.

Close the currently open file.

BasicScript:

Sub Close ()

PascalScript:

Procedure Close

JScript:

Function Close()

Related Topics: Open Function, ReadLine Function, WriteLine Procedure Example: See WriteLine Procedure.

CompareText Function

Return the result of comparing two text strings.

BasicScript:

Function CompareText (ByVal *s1* as String, ByVal *s2* as String) as Integer

PascalScript:

Function CompareText (*s1*, *s2* : String): Integer

JScript:

Function CompareText (*s1*, *s2*)

Copy Function

Return a substring of a specified string (Mid).

BasicScript:

Function Copy (ByVal *s* as String, ByVal *from* as Integer, ByVal *count* as Integer) as String

PascalScript:

Function Copy (*s* : String; *from*, *count* : Integer) : String

JScript:

Function Copy (*s*, *from*, *count*)

Copy returns a String.

The Copy function has these parts:

Part	Description
s	String expression from which another string is created.
fro	The *from* argument is a long expression that indicates the character

m	position in *s* at which the part to be taken begins.
count	The *count* is a long expression that indicates the number of characters to return.

Related Topics: Mid Function, Left Function, Len Function, Right Function, Mid Function

CopyFile Function

Copy a file's contents to another file.

BasicScript:

Function CopyFile (ByVal *SourceFile* as String, ByVal *DestFile* as String) as Boolean

PascalScript:

Function CopyFile (*SourceFile* : String, *DestFile* : String) : Boolean

JScript:

Function CopyFile (*SourceFile*, *DestFile*)

Returns True if successful, else False.

CopyToClipboard Procedure

Applies to: TTermScreen Class.

Copy the marked text to the Windows clipboard. This subroutine must be preceded by a **MarkBlock** subroutine.

BasicScript:

Sub CopyToClipboard ()

PascalScript:

Procedure CopyToClipboard

JScript:

Function CopyToClipboard()

Related Topics: MarkBlock , PasteFromClipboard

Cos Function

Return the cosine of an angle.

BasicScript:

Function Cos (ByVal *e* as Extended) as Extended

PascalScript:

Function Cos (*e* : Extended) : Extended

JScript:

Function Cos(*e*)

BasicScript Example:

```
Msg = ""
For I =1 To 2
  Msg = Msg & FloatToStr(Cos(I)) & ", "  ' Cos function call
  J=Cos(I)
  MsgBox (FloatToStr(J))
Next
MsgBox (Msg)                             ' Display results.
```

Create Function

Applies to: TDialogForm Class and TOpenDialog Class.

Create an instance of a class object. Use Create in PascalSripts; New in BasicScripts and JScripts.

BasicScript:

Function New (ByVal *Owner* as TObject) as Variant

PascalScript:

Function Create (*Owner* : TObject) : Variant

JScript:

Function New (*Owner*)

Example BasicScript/JScript:

```
MyDialog = New TDailogForm(Self)
```

Example PascalScript:

```
MyDialog := TDialogForm.Create(Self)
```

Related Topics: Free Procedure, LoadForm Function, SetVariable Procedure, GetVariable Function, ShowForm Function, ClearForm Procedure, PrintForm Procedure

The following are examples of using the TDialogObject in an eXpress Script:

BasicScript:

```
df = New TDialogForm(Self)
Try
df.LoadForm(ScriptFolder + "\\NEWDIALOGTEST.bfm", true)
rslt = df.ShowForm
If rslt = mrOk Then
  MsgBox("You selected:" + df.Edit_1.Text, mb_iconinformation, "Result")
Else
  MsgBox("Cancelled", mb_iconinformation, "Result")
End If
Finally
  df.Free
End Try
```

PascalScript:

```
Var Df : variant;
Var Rslt : integer;
df = TDialogForm.Create(Self)

begin
Try
df.LoadForm(ScriptFolder + '\NEWDIALOGTEST.bfm', true);
rslt := df.ShowForm;
If rslt = mrOk Then
  MsgBox('You selected:' + df.Edit_1.Text, mb_iconinformation, 'Result')
Else
  MsgBox('Cancelled', mb_iconinformation, 'Result');
Finally
  df.Free;
End Try
End.
```

JScript:

```
Var df, rskt

df = New TDialogForm(Self)
Try
df.LoadForm(ScriptFolder + "\NEWDIALOGTEST.bfm", true)
rslt = df.ShowForm
If rslt = mrOk Then
  MsgBox("You selected:" + df.Edit_1.Text, mb_iconinformation, "Result")
Else
  MsgBox("Cancelled", mb_iconinformation, "Result")
End If
Finally
  df.Free
End Try
```

CreateFolder Function

Create a file folder.

BasicScript:

Function CreateFolder (ByVal *FolderName* as String) as Boolean

PascalScript:

Function CreateFolder (*FolderName* : String) : Boolean

JScript:

Function CreateFolder (*FolderName*)

Returns True if successful, else False.

CreateOleObject Function

Create an OLE automation object.

BasicScript:

Function CreatOleObject (ByVal *ClassName* as String) as Variant

PascalScript:

Function CreateOleObject (*ClassName* : String) : Variant

JScript:

Function CreateOleObject (*ClassName*)

The *ClassName* parameter has the following format:

"*AppName.ObjectType*"

The *class* parameter has the following parts:

Part	Description
AppName	Name of the application providing the object.
ObjectType	Type or class of object to create.

BasicScript Example:

```
  #Language BasicScript
  'This script will start an instance of Microsoft Word and will automatically
  ' load the contents of the screen into the document.  This script can be
  ' customized to take only certain portions of the screen data or to customize
  ' a letter around the data to make it more usful to your site or organization.
    'VARIABLES
      Dim MSWordObj
      Dim x

      'Create Word Basic Object
    MSWordObj = CreateOleObject("Word.Basic")

      'Create the New Document and Other Settings

      'Start a New Document
      MSWordObj.FileNewDefault
      'View the Current Page
      MSWordObj.ViewPage
      'Insert a Paragraph Break
      MSWordObj.InsertPara

      MSWordObj.Font("Times New Roman")
      MSWordObj.FontSize(11)
      MsWordObj.Insert("This is a sample Word Script." + Chr(13))
      MsWordObj.Insert("Your screen contents will display below: " + Chr(13) + Chr(13))

      'Loop through each Row and print contents to document using fixed font
      For x = 1 To 24
        MSWordObj.Font("Courier New")
        MSWordObj.FontSize(9)
         MsWordObj.Insert(TermScreen.GetScreenText(1,x,80) + Chr(13))
      Next

      'Show the Word Application
       MSWordObj.AppShow
```

Date Function

Return the current system date.

BasicScript:

Function Date () as TDateTime

PascalScript:

Function Date () : TDateTime

JScript:

Function Date ()

Related Topics: Format Function, Now Function

DateTimeToStr Function

Convert date and time to a string.

BasicScript:

Function DateTimeToStr (ByVal *e* as Extended) as String

PascalScript:

Function DateTimeToStr (*e* : Extended) : String

JScript:

Function DateTimeToStr (*e*)

DateToStr Function

Convert date to a string.

BasicScript:

Function DateToStr (ByVal *e* as Extended) as String

PascalScript:

Function DateToStr (*e* : Extended) : String

JScript:

Function DateToStr (*e*)

DayOfWeek Function

Return the day of the week using a specified date.

BasicScript:

Function DayOfWeek (ByVal *aDate* as DateTime) as Integer

PascalScript:

Function DayOfWeek (*aDate* : DateTime) : Integer

JScript:

Function DayOfWeek (*aDate*)

DaysInMonth Function

Return the number of days in a specified month.

BasicScript:

Function DaysInMonth (ByVal *nYear* as Integer, ByVal *nMonth* as Integer) as Integer

PascalScript:

Function DaysInMonth (*nYear*, *nMonth* : Integer) : Integer

JScript:

Function DaysInMonth (*nYear*, *nMonth*)

Dec Procedure

Decrement an integer variable.

BasicScript:

Sub Dec (ByRef *i* as Integer, ByVal *decr* as Integer = 1)

PascalScript:

Procedure Dec (var *i* : Integer; *decr* : Integer = 1)

JScript:

Function Dec (*i* , *decr* as Int = 1)

DecodeDate Procedure

Return the year, month and day values for a date.

BasicScript:

Sub DecodeDate (ByVal *Date* as TDateTime, ByRef *Year* as Integer, ByRef *Month* as Integer, ByRef *Day* as Integer)

PascalScript:

Procedure DecodeDate (*Date* : TDateTime; var *Year*, *Month*, *Day* : Integer)

JScript:

Function DecodeDate (*Date*, *Year*, *Month*, *Day*)

DecodeTime Procedure

Return the hours, minutes, seconds and milliseconds of a time.

BasicScript:

Sub DecodeTime (ByVal *Time* as TDateTime, ByRef *Hour* as Integer, ByRef *Min* as Integer, ByRef *Sec* as Integer ByRef *MSec* as Integer)

PascalScript:

Procedure DecodeTime (*Time* : TDateTime; var *Hour*, *Min*, *Sec*, *MSec* : Integer)

JScript:

Function DecodeTime (*Time*, *Hour*, *Min*, *Sec*, *MSec*)

DeleteFile Function

Delete the specified file.

BasicScript:

Function DeleteFile (ByVal *FileName* as String) as Boolean

PascalScript:

Function DeleteFile (*FileName* : String) : Boolean

JScript:

Function DeleteFile (*FileName*)

Returns True if successful, else False.

DeleteStr Procedure

Return a string result from deleting a part of a string.

BasicScript:

Sub DeleteStr (ByRef *CurrStr* as String, ByVal *FromPos* as Integer, ByVal *Count* as Integer)

PascalScript:

Procedure DeleteStr (var *CurrStr* : String; *FromPos*, Count : *Integer*)

JScript:

Function DeleteStr (*CurrStr*, *FromPos*, *Count*)

Deletes positions specified by *FromPos* and *Count* from string specified by *CurrStr*.

DoTerminalKey Procedure

Applies to: TTermScreen Class.

Issue any of the supported T27 or UTS keystrokes. Note: The supported keystrokes are dependant upon which emulator is being used. UTS eXpress Enterprise only supports UTS keys, while T27 eXpress Enterprise supports only T27 keys.

BasicScript:

Sub DoTerminalKey (ByVal *key* as Integer) as Integer

PascalScript:

Procedure DoTerminalKey (*key* : Integer)

JScript:

Function DoTerminalKey (*key*)

The *key* parameter is an integer expression representing the specific key to be issued. The *key* may be specified as an Integer or Constant.

T27 Constants/key integers:

Constant	Integer
TK_ARROWDN	249
TK_ARROWLEFT	247
TK_ARROWRIGHT	248
TK_ARROWUP	246
TK_BACKSPACE	8
TK_BACKTAB	196
TK_BOUND	218
TK_CARRIAGERTN	13
TK_CLRALLVTAB	16442
TK_CLREOL	134
TK_CLREOP	135
TK_CLRFORMS	159
TK_CLRHOME	128
TK_COPY	16432
TK_CTRL	164
TK_CUT	16431
TK_DBLZERO	234
TK_DELCHAR	132
TK_DELCHARPAGE	16425
TK_DELLINE	133
TK_HOME	174
TK_INSCHAR	130
TK_INSCHARPAGE	16424
TK_INSLINE	131
TK_LOCAL	168
TK_LOCKCTRL	165
TK_LOGICALEOL	16415
TK_MARK	217
TK_MOVELINEDOWN	138
TK_MOVELINEUP	139
TK_NEXTPAGE	253
TK_PASTE	16434
TK_PREVPAGE	252
TK_PRINTALL	157
TK_PRINTUNPROT	156
TK_RECALL	214
TK_RECEIVE	170
TK_ROLLDN	136
TK_ROLLUP	137
TK_SETFORMS	158
TK_SPECIFY	166
TK_STORE	213
TK_TAB	198
TK_TOGGLEFORMS	141
TK_TOGGLETAB	16441
TK_TRANSMIT	172
TK_TRANSMITLINE	16428
TK_TRIPZERO	236
TK_UPPERONLYON	210
TK_UPPERONLYOFF	211
TK_WRITEESC	16426
TK_WRITEETX	3
TK_WRITEGS	16427

UTS Constants/key integers:

Constant	Integer
UK_BACK_SPACE	95
UK_CURSOR_DOWN	6

UK_CURSOR_LEFT	7
UK_CURSOR_RETURN_KEY	32
UK_CURSOR_RIGHT	8
UK_CURSOR_TO_END_LINE	66
UK_CURSOR_TO_HOME	23
UK_CURSOR_TO_START_LINE	65
UK_CURSOR_UP	9
UK_DELETE_IN_DISPLAY	11
UK_DELETE_IN_LINE	12
UK_DELETE_LINE	10
UK_ERASE_CHAR	67
UK_ERASE_DISPLAY	14
UK_ERASE_TO_END_DISPLAY	15
UK_ERASE_TO_END_FIELD	16
UK_ERASE_TO_END_LINE	17
UK_FKEY_1	43
UK_FKEY_2	44
UK_FKEY_3	45
UK_FKEY_4	46
UK_FKEY_5	47
UK_FKEY_6	48
UK_FKEY_7	49
UK_FKEY_8	50
UK_FKEY_9	51
UK_FKEY_10	52
UK_FKEY_11	53
UK_FKEY_12	54
UK_FKEY_13	55
UK_FKEY_14	56
UK_FKEY_15	57
UK_FKEY_16	58
UK_FKEY_17	59
UK_FKEY_18	60
UK_FKEY_19	61
UK_FKEY_20	62
UK_FKEY_21	63
UK_FKEY_22	64
UK_INSERT_IN_DISPLAY	25
UK_INSERT_IN_LINE	26
UK_INSERT_LINE	24
UK_KEYBOARD_UNLOCK	27
UK_LINE_DUP	28
UK_MSG_WAIT	29
UK_PRINT_KEY	30
UK_PRINT_ENTIRE_SCREEN	69
UK_SOE	3
UK_TAB_BACK	33
UK_TAB_FORWARD	34
UK_TAB_SET	35
UK_TRANSMIT_KEY	36

Draw Procedure

Applies to: TCanvas Class.

Draw a graphic at the specified x, y coordinates.

BasicScript:

Sub Draw (ByVal *x* as Integer, ByVal *y* as Integer, ByVal *Graphic* as TGraphic)

PascalScript:

Procedure Draw (x : Integer, *y* : Integer, *Graphic* : TGraphic)

JScript:

Function Draw (*x*, *y*, *Graphic*)

Related Topics: Ellipse Procedure, LineTo Procedure, MoveTo Procedure, Rectangle Procedure, RoundRectangle Procedure, StretchDraw Procedure, TextHeight Function, TextOut Procedure, TextWidth Function

DrawRect Procedure

Applies to: TXSPrint Class.

Draw a rectangle using the specified pixel coordinates.

BasicScript:

Sub DrawRect (ByVal *Left* as Integer, ByVal *Top* as Integer, ByVal *Right* as Integer, ByVal *Bottom* as Integer)

PascalScript:

Procedure DrawRect (*Left* : Integer; *Top* : Integer; *Right* : Integer; *Bottom* : Integer)

JScript:

Function DrawRect (*Left*, *Top*, *Right*, *Bottom*)

Related Topics: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, LineTo Procedure, NewPage Procedure

Ellipse Procedure

Applies to: TCanvas Class.

Draw an Ellipse within the bounds specified by x1, y1, x2 and y2 using the current pen and brush.

BasicScript:

Sub Ellipse (ByVal *x1* as Integer, ByVal *y1* as Integer, ByVal *x2* as Integer, ByVal *y2* as Integer)

PascalScript:

Procedure Ellipse(*x1* : Integer; *y1* : Integer; *x2* : Integer; *y3* : Integer)

JScript:

Function Ellipse (*x1*, *y1*, *x2*, *y2*)

Related Topics: Draw Procedure, LineTo Procedure, MoveTo Procedure, Rectangle Procedure, RoundRectangle Procedure, StretchDraw Procedure, TextHeight Function, TextOut Procedure, TextWidth Function

EncodeDate Function

Return a data from specified year, month and day.

BasicScript:

Function EncodeDate (ByVal *Year* as Integer, ByVal *Day* as Integer, ByVal *Year* as Integer) as TDateTime

PascalScript:

Function EncodeDate (*Year*, *Month*, *Day*: Integer): TDateTime

JScript:

Function EncodeDate (*Year*, *Month*, *Day*)

EncodeTime Function

Return a time from specified hour, minute, second and millisecond.

BasicScript:

Function EncodeTime (ByVal *Hour* as Integer, ByVal *Min* as Integer, ByVal *Sec* as Integer, ByVal *MSec* as Integer) as TDateTime

PascalScript:

Function EncodeTime (*Hour*, *Min*, *Sec*, *MSec*: Integer): TDateTime

JScript:

Function EncodeTime (*Hour*, *Min*, *Sec*, *MSec*)

EndDoc Procedure

Applies to: TXSPrint Class and TXSLinePrinter Class.

Ends the current printer document and send it to the printer.

BasicScript:

Sub EndDoc ()

PascalScript:

Procedure EndDoc

JScript:

Function EndDoc()

Related Topics: BeginDoc Procedure, NewPage Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

EnterText Procedure

Applies to: TTermScreen Class.

Enter the specified value at the current cursor position on the screen.

BasicScript:

Sub EnterText (ByVal *Value* as String)

PascalScript:

Procedure EnterText (*Value* : String)

JScript:

Function EnterText (*Value*)

The *Value* parameter is any valid string expression.

Example:

(See GetScreenLine Function).

EnterTextFromPrompt Procedure

Applies to: TTermScreen Class.

Enter a prompt string at the current cursor position.

BasicScript:

Sub EnterTextFromPrompt (*Prompt* : String)

PascalScript:

Procedure EnterTextFromPrompt (*Prompt* : String)

JScript:

Function EnterTextFromPrompt (*Prompt* : String)

Execute Function

Applies to: TOpenDialog Class and TFontDialog Class.

Execute (show) an instance of a class object. Must be preceded by a Create (New).

BasicScript:

Function Execute () as Boolean

PascalScript:

Function Execute () : Boolean

JScript:

Function Execute ()

BasicScript Example:

```
Dim Fd
```

```
Fd = New TOpenDialog(Self) 'Create an instance
Fd.InitialDir = "C:\\MyFolder"
Fd.DefaultExt = "txt"
Fd.Filter = "Text files (*.txt)|*.txt|All file types (*.*)|*.*"
Fd.Title = "Open File Dialog Example"
If Fd.Execute then
  MsgBox("You selected file: " + fd.FileName)
Else
  MsgBox("Cancelled")
End If
Delete Fd ' Release instance
```

ExecuteProgram Function

Execute another application.

BasicScript:

Function ExecuteProgram (Byval *ExeFile* as String, ByVal *Parameters* as String = "", ByVal *WaitForComp* as Integer = 0) as Boolean

PascalScript:

Function ExecuteProgram (*ExeFile* : String, *Parameters* : String = "", *WaitForComp* : Integer = 0) : boolean

JScript:

Function ExecuteProgram (*ExeFile*, *Parameters*, *WaitForComp*)

ExtractFileExt Function

Get the extension of a file name. The period (.) is included; e.g., ".TXT".

BasicScript:

Function ExtractFileExt (ByVal *FileName* as String) as String

PascalScript:

Function ExtractFileExt (*FileName* : String) : String

JScript:

Function ExtractFileExt (*FileName*)

ExtractFileName Function

Get the file name portion (including extension) of a file reference — excludes the path.

BasicScript:

Function ExtractFileName (ByVal *FileName* as String) as String

PascalScript:

Function ExtractFileName (*FileName* : String) : String

JScript:

Function ExtractFileName (*FileName*)

ExtractFilePath Function

Get the path portion of a file name reference.

BasicScript:

Function ExtractFilePath (ByVal *FileName* as String) as String

PascalScript:

Function ExtractFilePath (*FileName* : String) : String

JScript:

Function ExtractFilePath (*FileName*)

Exp Function

Returns the base of the natural log raised to a power.

BasicScript:

Function Exp (ByVal *X* as Extended) as Extended

PascalScript:

Function Exp (*X*: Extended): Extended

JScript:

Function Exp (*X*)

BasicScript Example:

```
' Exp(x) is e ^x so Exp(1) is e ^1 or e.
Dim Msg, ValueOfE              ' Declare variables.
ValueOfE = Exp(1)              ' Calculate value of e.
Msg = "The value of e is " & ValueOfE
MsgBox (Msg)                   ' Display message.
```

FileExists Function

Check for the existence of a file.

BasicScript:

Function FileExists (ByVal *FileName* as String) as Boolean

PascalScript:

Function FileExists (*FileName* : String) : Boolean

JScript:

Function FileExists (*FileName*)

Returns True if successful, else False.

FloatToStr Function

Convert a floating-point value to a string.

BasicScript:

Function FloatToStr (ByVal *e* as Extended) as String

PascalScript:

Function FloatToStr (*e*: Extended): String

JScript:

Function FloatToStr (*e*)

BasicScript Example:

```
Msg = ""
For I =1 To 2
  Msg = Msg & FloatToStr(Cos(I)) & ", "  ' FloatToStr function call
  J=Cos(I)
  MsgBox (FloatToStr(J))
Next
MsgBox (Msg)                           ' Display results.
```

FolderExists Function

Check for the existence of a folder.

BasicScript:

Function FolderExists (ByVal *FolderName* as String) as Boolean

PascalScript:

Function FolderExists (*FolderName* : String) : Boolean

JScript:

Function FolderExists (*FolderName*)

Returns True if successful, else False.

Format Function

Return a formatted string assembled from a format string and an array of arguments.

BasicScript:

Function Format (ByVal *Format* as String, ByVal *Args* as Array) as String

PascalScript:

Function Format (*Format*: String; *Args*: Array) : String

JScript:

Function Format (*Format*, *Args*)

The Format function formats the series of arguments in an open (untyped) array.

Format is the format string. For information on the format strings, see Format Strings, described in this topic.

Args is an array of arguments to apply to the format specifiers in *Format*.

Format returns the results of applying the arguments in Args to the format string Format.

Format Strings

Format strings specify required formats to general-purpose formatting routines. Format strings passed to the string formatting routines contain two types of objects — literal characters and format specifiers. Literal characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Format specifiers have the following form:

"%" [*index* ":"] ["-"] [*width*] ["." *prec*] *type*

A format specifier begins with a % character. After the % comes the following elements, in this order:

An optional argument zero-offset index specifier (that is, the first item has index 0), [*index* ":"]

An optional left justification indicator, ["-"]

An optional width specifier, [*width*]

An optional precision specifier, ["." *prec*]

The conversion type character, *type*

The following table summarizes the possible values for *type*:

Value	Meaning
d	Decimal. The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.
u	Unsigned decimal. Similar to 'd' but no sign is output.
e	Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.
f	Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string—a default of 2 decimal digits is assumed if no precision specifier is present.
g	General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.
n	Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format,

except that the resulting string contains thousand separators.

m Money. The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator, and CurrencyDecimals global variables or their equivalent in a TFormatSettings data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals global variable or its TFormatSettings equivalent.

p Pointer. The argument must be a pointer value. The value is converted to an 8 character string that represents the pointers value in hexadecimal.

s String. The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.

x Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Conversion characters may be specified in uppercase as well as in lowercase—both produce the same results.

Index, width, and precision specifiers can be specified directly using decimal digit string (for example "%10d"), or indirectly using an asterisk character (for example "%*.*f"). When using an asterisk, the next argument in the argument list (which must be an integer value) becomes the value that is actually used. For example,

```
Format ('%*.*f', [8, 2, 123.456]);
```

is equivalent to:

```
Format ('%8.2f', [123.456]);
```

Similarly:

```
TVarRec args[3] = {8,2,123.456};
Format ("%*.*f", args, 2);
```

is equivalent to:

```
TVarRec args[1] = {123.456};
Format ("%8.2f", args, 0);
```

A *width* specifier sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width. The default is to right-justify the result by adding blanks in front of the value, but if the *format* specifier contains a left-justification indicator (a "-" character preceding the width specifier), the result is left-justified by adding blanks after the value.

An *index* specifier sets the current argument list index to the specified value. The *index* of the first argument in the argument list is 0. Using *index* specifiers, it is possible to format the same argument multiple times. For example "Format('%d %d %0:d %1:d', [10, 20])" produces the string '10 20 10 20'.

Note: Setting the *index* specifier affects all subsequent formatting. That is, Format('%d %d %d %0:d %d', [1, 2, 3, 4]) returns '1 2 3 1 2', not '1 2 3 1 4'. To get the latter result, you must use Format('%d %d %d %0:d %3:d', [1, 2, 3, 4])

Pascal Example:

```
Var
  x : integer;
  f : Extended;
  s : String;

Begin
  f := 3.14189;
  x := 35;
  s := 'This is string';

  MsgBox(Format('Formated float: %f, formated integer: %2.2d and a string
(%s)', [f, x, s]));

End.
```

BasicScript Example:

```
f = 3.14189
x = 35
s = "This is string"
```

```
MsgBox(Format("Formated float: %f, formated integer: %2.2d and a string
(%s)", [f, x, s]))
```

FormatDateTime Function

Format a TDateTime value.

BasicScript:

Function FormatDateTime (ByVal *Format* as String, ByVal *DateTime* as TDateTime) as String

PascalScript:

Function FormatDateTime (*Format*: String; *DateTime*: TDateTime) : String

JScript:

Function FormatDateTime (*Format*, *DateTime*)

FormatDateTime formats the TDateTime value given by *DateTime* using the format given by *Format*. See the table below for information about the supported format strings.

If the string specified by the *Format* parameter is empty, the TDateTime value is formatted as if a 'c' format specifier had been given.

Date-Time Format Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d") simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	Displays the date using the format given by the ShortDateFormat global variable.
dddddd	Displays the date using the format given by the LongDateFormat global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).

hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).
t	Displays the time using the format given by the ShortTimeFormat global variable.
tt	Displays the time using the format given by the LongTimeFormat global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator global variable.
:	Displays the time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

Pascal Examples:

```
// The following example uses FormatDateTime to set the string
// variable S to a sentence indicating a meeting time in 3
// hours. The sentence has the form "The meeting is on
// Wednesday, February 15, 1995 at 2:30 PM."
//
procedure TForm1.Button1Click(Sender: TObject);
var S : string;
begin
  S := FormatDateTime('"The meeting is on " dddd, mmmm d, yyyy, " at " hh:mm
AM/PM', Now + 0.125);
  Label1.Caption := S;
end;
```

FormatFloat Function

Format a floating-point value.

BasicScript:

Function FormatFloat (ByVal *Format* as String, ByVal *Value* as Extended) as String

PascalScript:

Function FormatFloat (*Format*: String; *Value*: Extended) : String

JScript:

Function FormatFloat (*Format*, *Value*)

FormatFloat formats the floating-point value given by *Value* using the format string given by *Format*. The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are

	ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator global variable or its TFormatSettings equivalent.
,	Thousand separator. If the format string contains one or more ',' characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of ',' characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator global variable or its TFormatSettings equivalent.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStr with the ffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

Pascal Example:

```
f := 21.34;
MyStr := 'Formated floating pound example result: ' + FormatFloat('####.00', f);
```

FormatMaskText Function

Return a string formatted using an edit mask.

BasicScript:

Function FormatMaskText (ByVal *EditMask* as String, ByVal *Value* as String) as String

PascalScript:

Function FormatMaskText (*EditMask* : String; *Value* : String) : String

JScript:

Function FormatMaskText (*EditMask*, *Value*)

Call FormatMaskText to apply the mask specified by the *EditMask* parameter to the text string specified by the *Value* parameter. The edit mask string consists of three fields with semicolons separating them. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of the mask are matched to characters in the *Value* parameter or are

inserted into the Value string. The third part of the mask is the character used to represent missing characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the returned string as leading blanks. If a ! character is not present, optional characters are represented in the returned string as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.
<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case present in the Value parameter.
\	The character that follows a \ character is a literal character. Use this character to use any of the mask special characters as a literal.
L	The L character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The l character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.
C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel, that character is substituted in the returned string.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel, that character is substituted in the returned string.
;	The ; character is used to separate the three fields of the mask.
_	The space (_) character automatically inserts spaces into the returned string.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters are inserted automatically if the second field of the mask is 0, or matched to characters in the *Value* parameter if the second field is any other value. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask are included in the *Value* parameter. For example, the mask for a telephone number with area code could be the following string:

(000)_000-0000;0;*

The 0 in the second field indicates that the Value parameter should consist of the 10 digits of the phone number, rather than the 14 characters that make up the final formatted string.

A 0 in the second field indicates that literals are inserted into the *Value* string, any other character indicates that they should be included.

The third field of the mask is the character that appears in the returned string for blanks (characters that do not appear in *Value*). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in the returned string.

BasicScript Example:

```
PhoneNo = "7706356350"
```

```
MsgBox("Formatted phone number: " + FormatMaskText("(000)_000-0000;0;*",
PhoneNo))
```

Frac Function

Return the fractional part of a numeric expression.

BasicScript:

Function Frac (By Val *X* as Extended) as Extended

PascalScript:

Function Frac (*X*: Extended) : Extended

JScript:

Function Frac (*X*)

Free Procedure

Applies to: TDialogForm Class.

This method disposes of the instance of the TDialogForm object. Once freed, an object must not be accessed, unless it is created again.

BasicScript:

Sub Free ()

PascalScript:

Procedure Free

JScript:

Function Free()

Related Topics: Create Function, LoadForm Function, SetVariable Procedure, GetVariable Function, ShowForm Function, ClearForm Procedure, PrintForm Procedure

Example: See Create Function.

GetFolderPath Function

Get the folder path of a Windows known folder.

BasicScript:

Function GetFolderPath (ByVal *CLSID* as Integer) as String

PascalScript:

Function GetFolderPath (*CLSID* : Integer) : String

JScript:

Function GetFolderPath (*CLSID*)

Valid *CLSID* integers and their constant equivalents are:

Integer	Constant
5	CSIDL_PERSONAL
26	CSIDL_APPDATA
28	CSIDL_LOCAL_APPDATA
32	CSIDL_INTERNET_CACHE
33	CSIDL_COOKIES
34	CSIDL_HISTORY
35	CSIDL_COMMON_APPDATA
36	CSIDL_WINDOWS
37	CSIDL_SYSTEM
38	CSIDL_PROGRAM_FILES
39	CSIDL_MYPICTURES
43	CSIDL_PROGRAM_FILES_COMMON
46	CSIDL_COMMON_DOCUMENTS

GetLastMsg Function

Applies to: TTermScreen Class.

In the UTS environment, this function retrieves the first 80 characters of the last message received (including any control sequences) from the host or communication system. In the T27 environment, this function retrieves the first 50 significant characters of the screen.

BasicScript:

Function GetLastMsg () as String

PascalScript:

Function GetLastMsg : String

JScript:

Function GetLastMsg()

The communication system may return multiple messages before control is returned to the script; therefore, only the last message is accessible by this function.

Since the message may contain control sequences, this function may not be very useful unless you are familiar with the handling of control sequences by the communications system. Consider using the GetScreenLine or GetScreenText functions.

Related Topics: GetScreenLine, GetScreenText, WaitForSpecificString, WaitForString

GetScreenAttribute Function

Applies to: TTermScreen Class.

Return Protected, Blink and Video Off attribute states and the specified column and row position.

BasicScript:

Function GetScreenAttribute (ByVal *Column* as Integer, ByVal *Row* as Integer) as Integer

PascalScript:

Function GetScreenAttribute (*Column* : Integer, *Row* : Integer) : Integer

JScript

Function GetScreenAttribute (*Column*, *Row*)

The *Column* and *Row* parameters are any integer expressions.

The attribute is a numeric expression containing a number equal to the sum of all required attributes.

The attribute may be checked using an Integer or Constant:

Attribute	Integer	Constant
Normal	0	ATTR_NORMAL
Start of Field	1	ATTR_FIELD
Tab Stop	2	ATTR_TAB
Data Field Changed	4	ATTR_CHANGED
Protected	8	ATTR_PROTECTED
Video Off	16	ATTR_VIDEO_OFF
Numeric Only Input	32	ATTR_NUMERIC
Alphabetic Only Input	64	ATTR_ALPHA
Blinking	128	ATTR_BLINK
Right Justified Data	256	ATTR_RIGHT
UTS Low Intensity	512	ATTR_LOWINT
Reverse Video	1024	ATTR_REV

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

Related Topic: GetScreenColor

BasicScript Example:

```
'This script selects all typed characters in the current field.

Dim CRow As Integer
Dim CCol As Integer
Dim SCol As Integer
Dim ECol As Integer
Dim FoundBlank As Boolean ' Integer
Dim s As Integer
Dim l As Integer
Dim EndDel As String

  ' Get the cursor row and column.
  CRow = TermScreen.CursorRow 'GetCursorRow()
```

```
  CCol = TermScreen.CursorColumn 'GetCursorCol()

  If (TermScreen.GetScreenAttribute(CCol, CRow) And 8) = 8 Then
                          ' Cursor must be in an unprotected area.
    Exit                  ' Exit Sub
  End If

  ' Find the end of the field (or end of the line)
  s = CCol
  ECol = 80
  While True              ' Do
    If (TermScreen.GetScreenAttribute(s, CRow) And 8) = 8 Then
      s = s - 1
      ECol = s
      Break               ' Exit Do
    End If
    If s >= 80 Then
      Break               ' Exit Do
    End If
    s = s + 1
  Wend                    ' Loop
  ' s now points to character before next protected region
  ' Work backward to the first non space (or beginning of the field)
  ' then get the end of the selection
  SCol = 1
  FoundBlank = False
  While true              ' Do
    If (TermScreen.GetScreenAttribute(s, CRow) And 8) = 8 Then
      SCol = s + 1
      Break               ' Exit Do
    End If
    If (FoundBlank = False) And (TermScreen.GetScreenText(s, CRow, 1) <> " ") Then
      FoundBlank = True
      ECol = s
    End If
    If s = 1 Then
      SCol = s
      Break               ' Exit Do
    End If
    s = s - 1
  Wend                    ' Loop
  ' Move the cursor the start of the field.
  TermScreen.SetCursor(SCol, CRow)
  ' Mark the selection
  TermScreen.MarkBlock(SCol, CRow, ECol, CRow)
  TermScreen.RefreshScreen
  ' Done.
End Sub
```

GetScreenColor Function

Applies to: TTermScreen Class.

Return a 2-digit hex number indicating the background and foreground color at the specified column and row position.

BasicScript:

Function GetScreenColor (ByVal *Column* as Integer, ByVal *Row* as Integer) as Integer

PascalScript:

Function GetScreenColor (*Column* : Integer, *Row* : Integer) : Integer

JScript:

Function GetScreenColor (*Column*, *Row*)

The *Column* and *Row* parameters are any integer expressions.

Colors are expressed as an index in the range 0 to 7. The first digit is the background color index and the second is foreground color index.

Colors are: 0 = black, 1 = Red, 2 = Green, 3 = Yellow, 4 = Blue, 5 = Magenta, 6 = Cyan, 7 = white.

For example, white text on a red background is 17 hexadecimal or 23 decimal.

Related Topic: GetScreenAttribute

GetScreenCount Function

Applies to: TTermScreen Class.

Returns the number of screens currently configured.

BasicScript:

Function GetScreenCount () as Integer

PascalScript:

Function GetScreenCount() : Integer

JScript:

Function GetScreenCount()

GetScreenLine Function

Applies to: TTermScreen Class.

Retrieve one logical line of the mapped terminal screen buffer. This function will retrieve any text within the specified area including protected and video off.

BasicScript:

Function GetScreenLine (ByVal *LineNumber* as Integer) : String

PascalScript:

Function GetScreenLine (*LineNumber* : Integer) : String

JScript:

Function GetScreenLine (*LineNumber*)

This function returns a String.

The *LineNumber* parameter is any integer expression, but must be within the range of 1 through the total number of lines of the terminal screen.

Related Topics: GetScreenText

GetScreenName Function

Applies to: TTermScreen Class.

Return the name of the screen at index. Index must be in the range 0 to Screen Count – 1.

BasicScript:

Function GetScreenName (ByVal *Index* as Integer) as String

PascalScript:

Function GetScreenName (*Index* : Integer) : String

JScript:

Function GetScreenName (*Index*)

BasicScript Example:

```
' Display a MsgBox containing the names
' of all configured screens indicating open screens.
c = TermScreen.GetScreenCount
s = ""
For x = 0 To c-1
  c = TermScreen.GetScreenName(x)
  If TermScreen.ScreenOpen(c) Then
    s = s + Chr(13) + c + "<OPEN>"
  Else
    s = s + Chr(13) + c
  End If
Next
MsgBox(s, 0, "Available Screens")
```

GetScreenText Function

Applies to: TTermScreen Class.

Retrieve a text string from the specified positions within the logical screen. This function will retrieve any text within the specified area including protected and video off.

BasicScript:

Function GetScreenText (ByVal *Col* as Integer, ByVal *Row* as Integer, ByVal *Len* as Integer) as String

PascalScript:

Function GetScreenText (*Col* : Integer, *Row* : Integer, *Len* : Integer) as String

JScript:

Function GetScreenText (*Col*, *Row*, *Len*)

The *Col*, *Row* and *Len* parameters are any integer expression.

If *row* is specified as -1, then *Col* is assumed to be the offset from the beginning of the logical screen buffer. For example:

```
GetScreenText (5, 2, 5)
```

Is the same as:

```
GetScreenText (85, -1, 5)
```

The above example assumes the screen has 80 columns.

Related Topics: GetScreenLine, SetScreenText

JScript Example:

```
// This example adds up the all the ammounts In the Total Charges column
// on the screen shown below. The total Is then displayed In a standard
// message box.
//-------------------------------------------------------------------------
//CUST6               ***************************
//                    **  Customer Detail List  **
//                    ***************************
//
//Account: 215183000        Tab To Desired Order For Order Details
//
//                    Order Id                 Total Charges
//                   01372030002                     131.93
//                   01345700102                     179.90
//                   01345700002                    5162.20
//                   01124032401                     555.00
//                   01124841601                     888.00
//                   01082906201                     555.00
//                   01193007701                    3996.00
//                   01094718401                    3108.00
//                   01115935001                     396.00
//                   01145334401                     222.00
//                   01093645801                    1776.00
//                   01074308701                    9879.00
//
//
//        Return | |          Exit | |
//-------------------------------------------------------------------------

Var Tot Extended;

  Tot = 0; // Initialize total

  For(Var x = 1; x < 12; x++)
    {
    s = Trim(TermScreen.GetScreenText(48, x+7, 9)); // Get amount from screen
    If (ValidFloat(s))  // Make sure its a valid float value
      Tot = Tot + StrToFloat(s);  // Add to total
    }

  MsgBox("All changes         ***" + Str(Tot), 0, "All Charges");
```

GetSessionVar Function

Applies to: TTermScreen Class.

Retrieve the current content of a global session variable. If the named session variable has not been set, an empty string is returned.

BasicScript:

Function GetSessionVar (ByVal *VarName* as String) as Variant

PascalScript:

Function SetSessionVar (*VarName* : String) : Variant

JScript:

Function GetSessionVar (*name*)

The *name* parameter is the string expression containing the session variable.

Related Topics: SetSessionVar

BasicSccript Example:

```
' This example totals up a column of numbers in the block marked by the user.
' A running total may be kept using a Session Variable

  If Not TermScreen.BlockMarked Then
    MsgBox("Please select the data to be totalled then run this script again.",
mb_IconInformation, "No Selection")
    Return
  End If

  Row = TermScreen.BlockStartRow
  l = TermScreen.BlockEndColumn - TermScreen.BlockStartColumn + 1
  If l < 1 Then
    MsgBox("Not enough columns selected.", mb_IconExcamation, "Columns")
    Return
  End If

  Tot = 0
  While Row <= TermScreen.BlockEndRow
    s = Trim(TermScreen.GetScreenText(TermScreen.BlockStartColumn, Row, l)) ' Get
amount from screen
    If (ValidFloat(s)) Then      ' Make sure its a valid float value
      Tot = Tot + StrToFloat(s)  ' Add to total
    Else
      MsgBox("Invalid numeric data encountered at row " + IntToStr(Row) + ".",
mb_IconExclamation, "Invalid Data")
      Return
    End If
    Inc(Row)
  Wend

  RunningTot = Tot + TermScreen.GetSessionVar("RunningTotal")

  answer = MsgBox(Format("This is your result: %9.2f, your running total is %9.2f.
" + Chr(13) + Chr(13) + "Do you want to clear the running total?", [Tot,
RunningTot]), mb_YesNo, "Result")

  If answer = IDYes Then
    TermScreen.SetSessionVar("RunningTotal", 0)
  Else
    TermScreen.SetSessionVar("RunningTotal", RunningTot)
  End If
```

GetTextHeight Function

Applies to: TXSPrint Class.

Return the pixel height of the specified text using the current printer and font settings.

BasicScript:

Function GetTextHeight (ByVal *Text* as String) as Integer

PascalScript:

Function GetTextHeight (*Text* : String) : Integer

JScript:

Function GetTextHeight (*Text*)

This function returns an integer. *Text* is a string expression.

Related Topics: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, NewPage Procedure, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

GetTextWidth Function

Applies to: TXSPrint Class.

Return the pixel width of the specified text using the current printer and font settings.

BasicScript:

Function GetTextWidth (ByVal *Text* as String) as Integer

PascalScript:

Function GetTextWidth (*Text* : String) : Integer

JScript:

Function GetTextWidth (*Text*)

This function returns an integer. *Text* is a string expression.

Related Topics: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, NewPage Procedure, TextOut Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

GetUserParam Function

Applies to: TTermScreen Class.

Retrieve user information for a calling script.

BasicScript:

Function GetUserParam (ByVal *Index* as Integer) as String

PascalScript:

Function GetUserParam (*Index* : Integer) : String

JScript:

Function GetUserParam (*Index*)

Return a string.

Index is the desired parameter number. Whenever a script is run, three parameters will be passed. Parameter 1 will always be the screen name. Parameter 2 will indicate whether this is a sign-on script (1 = Sign-on, 0 = other). The third parameter will be the toolbar button caption, menu item caption or an empty string if the script was started some other way.

The menu item caption will be passed as parameter 3 when a script is run from a menu. When the script is started from a toolbar button, the button caption will be passed.

For a script started by an action key sequence, parameter 3 will be set as follows:

KEY_*vvvsac*

Where:

vvvv is the virtual key code.

s is a Y or N indicating the SHIFT key was used.

a is a Y or N indicating the ALT key was used.

c is a Y or N indicating the CTRL key was used.

For example, <Ctrl> + A would be "KEY_065NNY", <Alt> + Enter would be "KEY_013NYN".

Parameter 3 will be an empty string when a script is run as an automatic sign-on script.

BasicScript Examples:

```
ButtonCap = TermScreen.GetUserParam(3)
ScreenName = TermScreen.GetUserParam(1)
```

GetVariable Function

Applies to: TDialogForm Class.

This method is used to retrieve the value of a global variable in a Dialog Form's action script. If the variable is not defined, the returned value will be an empty string.

BasicScript:

Function GetVariable (ByVal *VarName* as String) as Variant

PascalScript:

Function GetVariable (*VarName* : String) : Variant

JScript:

Function GetVariable (*VarName*)

Related Topics: Free Procedure, LoadForm Function, SetVariable Procedure, Create Function, ShowForm Function, ClearForm Procedure, PrintForm Procedure

HexToInt Function

Convert a string containing a hex value to a integer.

BasicScript:

Function HexToInt (ByVal *HexVal* as String) as Integer

PascalScript:

Function HexToInt (*HexVal* : String) : Integer

JScript:

Function HexToInt (*HexVal*)

HostIPAddress Function

Applies to: TTermScreen Class.

Get the IP Address of the host.

BasicScript:

Function HostIPAddress () as String

PascalScript:

Function HostIPAddress() : String

JScript:

Function HostIPAddress()

This function returns a string.

Inc Procedure

Increment an integer variable.

BasicScript:

Sub Inc (ByRef *i* as Integer, ByVal *incr* as Integer = 1)

PascalScript:

Procedure Inc (var *i* : Integer; *incr* : Integer = 1)

JScript:

Function Inc (*i*, *incr* as Int = 1)

InputBox Function

Return a string from an input dialog.

BasicScript:

Function InputBox (ByVal *Title* as String, ByVal *Prompt* as String, ByVal *DefautValue* as String = "") as String

PascalScript:

Function InputBox (*Title* : String, *Prompt* : String, *DefautValue* : String = "") : String

JScript:

Function InputBox (*Title*, *Prompt*, *DefautValue* as String = "")

The InputBox function has these parts:

Part	Description
Title	String expression displayed in the title bar of the dialog.
Prompt	String expression displayed as the message in the dialog box.

DefaultValue	String expression displayed in the textbox as the default response, if no other input is provided.

Related Topics: InputQuery Function

BasicScript Example:

```
L = 0
Do
  Answer = InputBox("Enter a value from 1 to 3.", "", "")
  If (Answer >= 1) and (Answer <= 3) Then
    L = 1                              ' Set to exit Do Loop
  Else
    Beep (MB_ICONQUESTION)             ' Beep if not in range
  End If
Loop While L = 0
MsgBox ("You entered a value in the proper range.")
```

InputQuery Function

Return a string from an input dialog. Similar to the InputBox function, but the user input is returned through the "Value" parameter, not the result.

BasicScript:

Function InputQuery (ByVal *Title* as String, ByVal *Prompt* as String; ByRef *Value* as String) as Boolean

PascalScript:

Function InputQuery (*Title*, *Prompt* : String; var *Value* : String) : Boolean

JScript:

Function InputQuery (*Title*, *Prompt*, *Value*)

The Boolean result is True, if the user hits OK; False, if Cancel. When True, the user's response (a string) is returned in the Value parameter.

The InputQuery function has these parts:

Part	Description
Title	String expression displayed in the title bar of the dialog.
Prompt	String expression displayed as the message in the dialog box.
Value	String expression displayed in the textbox as the default response if no other input is provided.

Related Topics: InputBox Function

BasicScript Example:

```
Tmp = "" ' Initialize tmp
If Not InputQuery("SignOn", "Enter your password", Tmp) Then
  MsgBox("SignOn Cancelled")
  Exit
End If
' Continue sign on process
```

Insert Procedure

Return a string resulting from inserting one string into another.

BasicScript:

Sub Insert (ByVal NewStr as String, ByRef CurrStr as String, ByVal pos as Integer)

PascalScript:

Procedure Insert (NewStr: String; var CurrStr: String; pos: Integer)

JScript:

Function Insert (NewStr, CurrStr, pos)

InStr Function (Pos)

Return the position of a substring within a string (Pos).

BasicScript:

Function InStr (ByVal *StartChar* as Integer = 1, ByVal *SubStr* as String, ByVal *StrVal* as String) as Integer

PascalScript:

Function InStr (*StartChar*: Integer = 1, *SubStr* : String; *StrVal* : String) : Integer

JScript:

Function InStr (*StartChar* as Int = 1, *SubStr*, *StrVal*)

Return the character position of the first occurrence of *SubStr* within *StrVal*.

The *StartChar* parameter is not optional and sets the starting point of the search within *StrVal*. The *StartChar* parameter must be a valid positive integer, no greater than 65,535.

The *StrVal* parameter is the string being searched and *SubStr* is the string for which we are looking.

The function returns the following values:

If:	InStr returns:
SubStr is found within *StrVal*	Position at which match is found
SubStr is not found	0
SubStr is zero-length	*StartChar*
SubStr is Null	Null
StrVal is zero-length	0
StrVal is Null	Null
StartChar > *SubStr*	0

Related Topics: Len Function, Pos Function

BasicScript Example:

```
B = "Good Bye"
A = InStr(2, "Bye", B)  ' Returns a 5
MsgBox (A)
C = InStr(3, "Bye", B)  ' Returns a 4
MsgBox (C)
```

Int Function

Return the integer part of a numeric expression.

BasicScript:

Function Int (ByVal *e as* Extended) as Integer

PascalScript:

Function Int (*e* : Extended) : Integer

JScript:

Function Int (*e*)

InToHex Function

Convert an integer value to a string containing its hex value.

BasicScript:

Function IntToHex (ByVal *i* as Integer, ByVal *Digits* as Integer = 4) as String

PascalScript:

Function IntToHex (*i* : Integer, *Digits* : Integer = 4) : String

JScript:

Function IntToHex (*i*, *Digits* as Int = 4)

InToStr Function

Convert an integer value to a string.

BasicScript:

Function IntToStr (ByVal *i* as Integer) as String

PascalScript:

Function IntToStr (*i* : Integer) : String

JScript:

Function IntToStr (*i*)

IsLeapYear Function

Determine Leap Year from a specified year.

BasicScript:

Function IsLeapYear (ByVal Year as Integer) as Boolean

PascalScript:

Function IsLeapYear (Year : Integer) : Boolean

JScript:

Function IsLeapYear (Year)

LCase Function

Returns the value of a string converted to lower case (LowerCase).

BasicScript:

Function LCase (ByVal *s* as String) as String

PascalScript:

Function LCase (*s* : String) : String

JScript:

Function LCase (*s*)

Related Topics: UCase Function, Lowercase Function

BasicScript Example:

```
' This example uses the LTrim and RTrim functions to strip leading
' and trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the
' use of nested function calls.

    MyString = "  <-Trim->  "            ' Initialize string
    TrimString = LTrim(MyString)         ' TrimString = "<-Trim->  "
    MsgBox ("|" & TrimString & "|")
    TrimString = LCase(RTrim(MyString))  ' TrimString = "  <-trim->"
    MsgBox ("|" & TrimString & "|")
    TrimString = LTrim(RTrim(MyString))  ' TrimString = "<-Trim->"
    MsgBox ("|" & TrimString & "|")      ' Using the Trim function
                                         ' alone achieves the same
                                         ' result.
    TrimString = UCase(Trim(MyString))   ' TrimString = "<-TRIM->"
    MsgBox ("|" & TrimString & "|")
```

Left Function

Return a string containing the specified number of characters from the left side of a string.

BasicScript:

Function Left (ByVal *StrVal* as String, ByVal *Count* as Integer) as String

PascalScript:

Function Left (*StrVal* : String, *Count* : Integer) : String

JScript:

Function Left (*StrVal*, *Count*)

The *StrVal* parameter is the string expression from which the leftmost characters are returned.

The *Count* parameter is the numeric expression indicating the number of characters that will be returned.

Related Topics: Len Function, Mid Function, Right Function

BasicScript Example:

```
Dim LWord, Msg, RWord, SpcPos, UsrInp                    ' Declare variables
Msg = "Enter two words separated by a space."
UsrInp = InputBox("Enter Two Words",Msg,"first second")  ' Get user input
MsgBox (UsrInp)
SpcPos = InStr(1, " ", UsrInp)                           ' Find space
```

```
If SpcPos Then
    LWord = Left(UsrInp, SpcPos - 1)                    ' Get left word
    RWord = Right(UsrInp, Len(UsrInp) - SpcPos)
                                                        ' Get right word
    Msg = "The first word you entered is <" & LWord & ">"
    Msg = Msg & RWord & "."
Else
    Msg = "You didn't enter two words."
End If
MsgBox (Msg)                                            ' Display message
MidTest = Mid("Mid Word Test", 4, 5)
MsgBox (MidTest)
```

Len Function

Return the number of characters in a string (Length).

BasicScript:

Function Len (ByVal *s* as String) as Integer

PascalScript:

Function Len (*s* : String) : Integer

JScript:

Function Len (*s*)

Related Topics: InStr

BasicScript Example:

```
A = "Fast"
StrLen = Len(A)     ' the value of StrLen is 4
MsgBox (StrLen)
```

Length Function

Return the number of characters in a string (Len).

BasicScript:

Function Length (ByVal *s* as String) as Integer

PascalScript:

Function Length (*s* : String) : Integer

JScript:

Function Length (*s*)

Related Topics: InStr

BasicScript Example:

```
A = "Fast"
StrLen = Length(A)     ' the value of StrLen is 4
MsgBox (StrLen)
```

LineSpace Procedure

Applies to: TXSLinePrinter Class.

Advance the line counter leaving one or more blank lines. The optional *Count* parameter indicates the number of lines to advance. If omitted, the *count* is defaulted to 1 line. *Count* is limited to 10 lines.

BasicScript:

Sub LineSpace (ByVal *Count* as Integer)

PascalScript:

Procedure LineSpace (*Count* : Integer)

JScript:

Function LineSpace (*Count*)

Related Topics: EndDoc Procedure, NewPage Procedure, PrintLine Procedure, BeginDoc Procedure, Abort Procedure

LineTo Procedure

Applies to: TXSPrint Class and TCanvas Class.

TXSPrint Class:

Draw a line on the current page from the current drawing x and y position to the specified x and y position. The line's width is determined by the PenWidth property.

TCanvas Class:

Draws a line using the current pen from the current x, y coordinates to the specified x, y coordinates.

BasicScript:

Sub LineTo (ByVal *x* as Integer, ByVal *y* as Integer)

PascalScript:

Procedure LineTo(*x* : integer; *y* : integer)

JScript:

Function LineTo (*x*, *y*)

Related Tipocs: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, NewPage Procedure, DrawRect Procedure, Draw Procedure, Ellipse Procedure, Rectangle Procedure, RoundRectangle Procedure, StretchDraw Procedure, TextHeight Function, TextWidth Function

Ln Function

Return the log base of X.

BasicScript:

Function Ln (ByVal *X* as Extended) as Extended

PascalScript:

Function Ln (*X* : Extended) : Extended

JScript:

Function Ln (*X*)

LoadForm Function

Applies to: TDialogForm Class.

This method loads a Dialog Form from the specified file created using the Dialog Form designer. Set Debug to True to have the Dialog Form actions execute in debug mode. It the file does not exist or is invalid, the result will be False.

BasicScript:

Function LoadForm (ByVal *FileName* as String, ByVal *Debug* as Boolean = False) as Boolean

PascalScript:

Function LoadForm (*FileName* : String, *Debug* : Boolean = False) : Boolean

JScript:

Function LoadForm (*FileName, Debug* as Boolean = False)

Related Topics: Free Procedure, Create Function, SetVariable Procedure, GetVariable Function, ShowForm Function, ClearForm Procedure, PrintForm Procedure

Example: See Create Function.

LoadScreen Procedure

Applies to: TTermScreen Class.

Load an entire screen/form from a file.

BasicScript:

Sub LoadScreen (ByVal *FileName* as String)

PascalScript:

Procedure LoadScreen (*FileName* : String)

JScript:

Function LoadScreen (*FileName*)

The *FileName* parameter is any string expression containing the file name of a previously saved screen/form file. Specific form loads can be assigned to function keys.

Related topic: SaveScreen Procedure

Example:

```
#Language BasicScript
Explicit
Dim dlg
  dlg = New TOpenDialog(Self)
  Try
    If dlg.execute Then
      If Not TermScreen.LoadScreen(dlg.FileName) Then
        MsgBox("Not loaded")
      Else
        MsgBox("Loaded")
      End If
    End If
  Finally
    Dlg.Free
  End Try
```

Lowercase Function

Returns the value of a string converted to lower case (LCase).

BasicScript:

Function Lowercase (ByVal *s* as String) as String

PascalScript:

Function Lowercase (*s* : String) : String

JScript:

Function Lowercase (*s*)

Related Topics: UCase Function,

BasicScript Example:

```
' This example uses the LTrim and RTrim functions to strip leading
' and trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the
' use of nested function calls

    MyString = "  <-Trim->  "               ' Initialize string
    TrimString = LTrim(MyString)            ' TrimString = "<-Trim->  "
    MsgBox ("|" & TrimString & "|")
    TrimString = Lowercase(RTrim(MyString)) ' TrimString = "  <-trim->"
    MsgBox ("|" & TrimString & "|")
    TrimString = LTrim(RTrim(MyString))     ' TrimString = "<-Trim->"
    MsgBox ("|" & TrimString & "|")         ' Using the Trim function
                                            ' alone achieves the same
                                            ' result.
    TrimString = Uppercase(Trim(MyString))  ' TrimString = "<-TRIM->"
    MsgBox ("|" & TrimString & "|")
```

LTrim Function

Return a string form a string trimmed of spaces from the left side.

BasicScript:

Function LTrim (ByVal *s* as String) as String

PascalScript:

Function LTrim (*s* : String) : String

JScript:

Function LTrim (*s*)

Related Topics: RTrim Function, Trim Function

Examples: See Trim Function

MakeString Function

Return a string of a specified length containing all spaces (Space).

BasicScript:

Function MakeString (ByVal *Length* as Integer, ByVal *FillChar* as Char = #32) as String

PascalScript:

Function MakeString (*Length* : Integer, *FillChar* : Char = #32) : String

JScript:

Function MakeString (*Length*, *FillChar* as Char = #32)

Related Topics: Trim Function,

Examples: See Trim Function

MarkBlock Procedure

Applies to: TTermScreen Class.

Mark a block of text on the screen to be subsequently copied to the Windows clipboard by the **CopyToClipboard** procedure.

BasicScript:

Sub MarkBlock (ByVal *SCol* as Integer, ByVal *SRow* as Integer, ByVal *Ecol* as Integer, ByVal *ERow* as Integer)

PascalScript:

Procedure MarkBlock (*SCol* : Integer, *SRow* : integer, *ECol* : Integer, *ERow* : Integer)

JScript:

Function MarkBlock (*SCol, SRow, ECol, ERow*)

SCol, SRow, ECol and *ERow* are numeric expressions indicating the boundaries of the block of screen text to be copied to the Windows clipboard.

Related Topics: CopyToClipboard , PasteFromClipboard

Mid Function

Return a substring of a specified string (Copy).

BasicScript:

Function Mid (ByVal *StrVal* as String, ByVal *StartPos* as Integer, ByVal *Count* as Integer) as String

PascalScript:

Function Mid(*StrVal* : String, *StartPos* : Integer; *Count* : Integer) : String

JScript:

Function Mid (*StrVal*, *StartPos*, *Count*)

Mid returns a String.

The Mid function has these parts:

Part	Description
StrVal	String expression from which another string is created.
StartPos	The *StartPos* argument is a long expression that indicates the character position in *s* at which the part to be taken begins.
Count	The *Count* is a long expression that indicates the number of characters to return.

Related Topics: Copy Function, Left Function, Len Function, Right Function

BasicScript Example:

```
Dim MidWord, Msg, TstStr, SpcPos1, SpcPos2, WordLen
TstStr = "Mid Function Demo"
SpcPos1 = InStr(1, " ", TstStr)              ' Find 1st space
SpcPos2 = InStr(SpcPos1 + 1, " ", TstStr)    ' Find 2nd space
WordLen = SpcPos2 - 1                        ' Get 2nd word length
MidWord = Mid(TstStr, SpcPos1 + 1, WordLen)  ' Get 2nd word
Msg = "The word in the middle of Title is '" & MidWord & "'."
```

```
MsgBox (Msg, 0, TstStr)
```

MoveTo Procedure

Applies to: TXSPrint Class and TCanvas Class.

TXSPrint Class:

Change the current page drawing position to the specified x and y coordinates.

TCanvas Class:

Moves the current x, y coordinates to the specifies x, y coordinates.

BasicScript:

Sub MoveTo (ByVal *x* as Integer, ByVal *y* as Integer)

PascalScript:

Procedure MoveTo (*x* : Integer; *y* : Integer)

JScript:

Function MoveTo (*x*, *y*)

Related Topics: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, NewPage Procedure, LineTo Procedure, DrawRect Procedure, Draw Procedure, Ellipse Procedure, LineTo Procedure, Rectangle Procedure, RoundRectangle Procedure, StretchDraw Procedure, TextHeight Function, TextWidth Function

MsgBox Function

Display a message in a dialog box and wait for the user to choose a button.

BasicScript:

Function MsgBox (ByVal Msg as String, ByVal Icon as Integer = 0, ByVal Title as String = "") as Integer

PascalScript:

Function MsgBox (Msg : String, Icon : Integer = 0, Title : String = "") : Integer

JScript:

Function MsgBox (Msg, Icon as Int = 0, Title as String = "")

MsgBox function returns a value indicating which button the user has chosen.

The *Msg* parameter is the string displayed in the dialog box as the message. The second and third parameters are optional and respectively designate the icon type of buttons and the title displayed in the dialog box.

The *Icon* is the sum of the values specifying the type of buttons to display, the icon style to use, the identity of the default button and the modality. The following illustrates the values and meaning of each group:

Constant	Value	Meaning
MB_OK	0	Display OK button only.
MB_OKCANCEL	1	Display OK and Cancel buttons.
MB_ABORTRETRYIGNORE	2	Display Abort, Retry and Ignore buttons.
MB_YESNOCANCEL	3	Display Yes, No and Cancel buttons.
MB_YESNO	4	Display Yes and No buttons.
MB_RETRYCANCEL	5	Display Retry and Cancel buttons.
————————	——	————————
MB_ICONSTOP	16	 Display:
MB_ICONQUESTION	32	 Display:
MB_ICONEXCLAMATION	48	 Display:
MB_ICONINFORMATION	64	 Display:
————————	——	————————
MB_DEFBUTTON1	0	First button is default.

MB_DEFBUTTON2	256	Second button is default.
MB_DEFBUTTON3	512	Third button is default.
MB_APPLMODAL	0	Application modal. The user must respond to the message box before continuing work in the current application.
MB_SYSTEMMODAL	4096	System modal. All applications are suspended until the user responds to the message box.

The first group of values (0-5) describes the number and type of buttons displayed in the dialog box. The second group (16, 32, 48, and 64) describes the icon style. The third group (0, 256 and 512) determines which button is the default. The fourth group (0 and 4096) determines the modality of the message box. When adding numbers to create a final value for the argument type, use only one number from each group. If omitted, the default value for type is zero.

The *Title* parameter is a string expression displayed in the title bar of the dialog box. If you omit the argument title, MsgBox has no default title.

The value returned by the MsgBox function indicates which button has been selected, as shown below:

Constant	Value	Meaning
IDOK	1	OK button selected.
IDCANCEL	2	Cancel button selected.
IDABORT	3	Abort button selected.
IDRETRY	4	Retry button selected.
IDIGNORE	5	Ignore button selected.
IDYES	6	Yes button selected.
IDNO	7	No button selected.

If the dialog box displays a Cancel button, pressing the Esc key has the same effect as choosing Cancel.

BasicScript Example:

The following example uses MsgBox to display a "close without saving" message in a dialog box with a Yes button, a No button and a Cancel button. The Yes button is the default response. The MsgBox function returns a value based on the button chosen by the user. The MsgBox statement uses that value to display a message that indicates which button was chosen.

```
Dim DgDef, Msg, Response, Title
Title = "MsgBox Sample Question"
Msg = "This is a sample of Close Without Saving?."
Msg = Msg & " Do you want to save changes?"
DgDef = MB_YESNOCANCEL + MB_ICONQUESTION + MB_DEFBUTTON1
Response = MsgBox(Msg, DgDef, Title)
If Response = IDYES Then
    Msg = "You chose Yes or pressed Enter."
ElseIf Response = IDCANCEL Then
    Msg = "You chose Cancel or pressed Esc."
Else
    Msg = "You chose No."
End If
MsgBox (Msg)
```

NameCase Function

Return the value of a string converted to name case (1st letter of words capitalized).

BasicScript:

Function NameCase (ByVal s as String)as String

PascalScript:

Function NameCase (s : String) : String

JScript:

Function NameCase (s)

New Function

See Create Function.

NewPage Procedure

Applies to: TXSPrint Class and TXSLinePrinter Class.

Insert a page break in the current printer document.

BasicScript:

Sub NewPage ()

PascalScript:

Procedure NewPage

JScript:

Function NewPage()

Related Topics: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, TextOut Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

Now Function

Return a date that represents the current date and time according to the settings in the computer's system date and time.

BasicScript:

Function Now () as TDateTime

PascalScript:

Function Now() : TDateTime

JScript:

Function Now()

The **Now** function returns a TDateTime data type containing a date and time that are stored internally.

Related Topics: Date Function, Format Function

BasicScript Example:

```
Dim Today
Today = Now
MsgBox (Today)  ' Produces today's date and time in the format of:
                '     mm/dd/yyyy hh:mm:ss
```

Open Function

Applies to: TXSTextFile Class.

Open the specified file in the specified file Mode. Open a file for input and output operations.

BasicScript:

Function Open (ByVal *FileName* as String, ByVal *FileMode* as TXSTextFileMode) as Boolean

PascalScript:

Function Open (*FileName* : String; *FileMode* : TXSTextFileMode) : Boolean

JScript:

Function Open (*FileName*, *FileMode*)

Available fileMode are:

fmRead	Open the file for reading
fmWrite	Open the file for writing
fmAppend	Open the file for writing and append new records to the end of the file when it already exists.

Related Topics: Close Procedure, ReadLine Function, WriteLine Procedure

Example: See WriteLine Procedure.

Ord Function

Return the integer value of a character (Asc).

BasicScript:

Function Ord (ByVal *ch* as Char) as Integer

PascalScript:

Function Ord (*ch* : Char) : Integer

JScript:

Function Ord (*ch*)

Related Topic: Asc Function

BasicScript Example:

```
Dim I, Msg                    ' Declare variables.
Msg = ""
For I = Ord("A") To Ord("Z")  ' From A through Z.
  Msg = Msg & Chr(I)          ' Create a string.
Next
MsgBox (Msg)                  ' Display results in Msg:
                              '   ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

PasteFromClipboard Procedure

Applies to: TTermScreen Class.

Paste the contents of the Windows clipboard to the current cursor position of the screen. This procedure is normally preceded by the **SetCursor** procedure.

BasicScript:

Sub PasteFromClipboard ()

PascalScript:

Procedure PasteFromClipboard

JScript:

Function PasteFromClipboard()

Related Topics: CopyToClipboard, MarkBlock, SetCursor

Pi Function

Return the value of pi.

BasicScript:

Function Pi () as Extended

PascalScript:

Function Pi () : Extended

JScript:

Function Pi()

Pos Function

Return the position of a substring within a string (InStr).

BasicScript:

Function Pos (ByVal *SubStr* as String, ByVal *s* as String) as Integer

PascalScript:

Function Pos (*SubStr* : String; *s* : String) : Integer

JScript:

Function Pos (*SubStr*, *s*)

Return the character position of the first occurrence of *SubStr* within *s*.

The parameter *s* is the string being searched and *SubStr* is the string for which we are looking.

The function returns the following values:

If:	InStr returns:
SubStr is found within *s*	Position at which match is found
SubStr is not found	0
SubStr is Null	Null
SubStr is zero-length	0
s is Null	Null

Related Topics: Len Function, InStr Function

BasicScript Example:

```
B = "Good Bye"
A = Pos("Bye", B)  ' Returns a 6
MsgBox (A)
```

PostAlert Procedure

Applies to: TTermScreen Class.

Post a message to the alert box.

BasicScript:

Sub PostAlert (ByVal *Title* as String, ByVal *Msg* as String, ByVal *Level* as Integer)

PascalScript:

Procedure PostAlert (*Title* : String, *Msg* : String, *Level* : Integer)

JScript:

Function PostAlert (*Title*, *Msg*, *Level*)

Title and *Msg* are string expressions. *Level* is a numeric expression equal to 0 (default) or set to one of the following:

Level	Constant
16	MB_ICONSTOP
32	MB_ICONQUESTION
48	MB_ICONEXCLAMATION

All screens globally update the alert box. Alert messages will be added to a list until the alert box is closed.

Each message will have Date/Time and Screen Name lines in front of the title. These two lines will be color coded per the icon color code.

PrintForm Procedure

Applies to: TDialogForm Class.

This method prints a copy of the current dialog form window.

BasicScript:

Sub PrintForm ()

PascalScript:

Procedure PrintForm

JScript:

Function PrintForm()

Related Topics: Free Procedure, LoadForm Function, SetVariable Procedure, GetVariable Function, ShowForm Function, ClearForm Procedure, Create Function

Examples:

The following examples use TDialogForm in an eXpress Script.

BasicScript

```
df = New TDialogForm(Self)
Try
df.LoadForm(ScriptFolder + "\\NEWDIALOGTEST.bfm", true)
rslt = df.ShowForm
If rslt = mrOk Then
  MsgBox("You selected:" + df.Edit_1.Text, mb_iconinformation, "Result")
Else
  MsgBox("Cancelled", mb_iconinformation, "Result")
End If
Finally
  df.Free
End Try
```

PascalScript

```
Var Df : variant;
Var Rslt : integer;
df = TDialogForm.Create(Self)
```

```
begin
Try
df.LoadForm(ScriptFolder + '\NEWDIALOGTEST.bfm', true);
rslt := df.ShowForm;
If rslt = mrOk Then
  MsgBox('You selected:' + df.Edit_1.Text, mb_iconinformation, 'Result')
Else
  MsgBox('Cancelled', mb_iconinformation, 'Result');
Finally
  df.Free;
End Try
End.
```

JScript

```
Var df, rskt

df = New TDialogForm(Self)
Try
df.LoadForm(ScriptFolder + "\NEWDIALOGTEST.bfm", true)
rslt = df.ShowForm
If rslt = mrOk Then
  MsgBox("You selected:" + df.Edit_1.Text, mb_iconinformation, "Result")
Else
  MsgBox("Cancelled", mb_iconinformation, "Result")
End If
Finally
  df.Free
End Try
```

PrintLine Procedure

Applies to: TXSLinePrinter Class.

Print a line of text using the *Text* parameter.

BasicScript:

Sub PrintLine (ByVal *Text* as String)

PascalScript:

Procedure PrintLine (*Text* : String)

JScript:

Function PrintLine (*Text)*

Related Topics: EndDoc Procedure, NewPage Procedure, BeginDoc Procedure, LineSpace Procedure, Abort Procedure

RaiseException Procedure

Cause a user-initiated exception to occur in a script.

BasicScript:

Sub RaiseException (ByVal *Param* as String)

PascalScript:

Procedure RaiseException (*Param* : String)

JScript:

Function RaiseException (*Param*)

Param is a string to be displayed in the exception message. Once an exception is raised, the script is terminated.

BasicScript Example:

```
...
If not ValidInt(MyStr) then
  RaiseException("That string does not contain a valid Integer")
End If
' The following will not be executed it the exception is raised
V = Val(MyStr)
...
```

Random Function

Return a random number in the range 0 <= X < 1. To initialize the random number generator, add a single call Randomize.

BasicScript:

Function Random () as Extended

PascalScript:

Function Random() : Extended

JScript:

Function Random()

Related Topic: Randomize Procedure

BasicScript Example:

```
Randomize ' Initialize random number generator
' Show 3 random numbers
For x = 1 To 3
  MsgBox(Format("Random number %d = %f", [x, Random]))
Next
```

Randomize Procedure

Randomize initializes the built-in random number generator with a random value (obtained from the system clock). The random number generator should be initialized by making a call to Randomize.

BasicScript:

Sub Randomize ()

PascalScript:

Procedure Randomize

JScript:

Function Randomize()

Do not combine the call to Randomize in a loop with calls to the Random function. Typically, Randomize is called only once, before all calls to Random.

Related Topic: Random Function

BasicScript Example:

```
Randomize ' Initialize random number generator
' Show 3 random numbers
For x = 1 To 3
  MsgBox(Format("Random number %d = %f", [x, Random]))
Next
```

ReadLine Function

Applies to: TXSTextFile Class.

Return the next line from the currently opened file.

BasicScript:

Function ReadLine (ByVal *ErrorStatus* as Integer) as String

PascalScript:

Function ReadLine (*ErrorStatus* : Integer) : String

JScript:

Function ReadLine (*ErrorStatus*)

The file's FileMode on the Open function must be fmRead.

If the read is successful, ErrorStatus will contain 0; otherwise, it will contain the system error code.

Related Topics: Open Function, Close Procedure, WriteLine Procedure

Example: See WriteLine Procedure.

Rectangle Procedure

Applies to: TCanvas Class.

Draws a rectangle bounded by the specified x1, y1, x2, y2 coordinates using the current brush and pen.

BasicScript:

Sub Rectangle (ByVal *x1* as Integer, ByVal *y1* as Integer, ByVal *x2* as Integer, ByVal *x2* as Integer)

PascalScript:

Procedure Rectangle (*x1* : Integer; *y1* : Integer; *x2* : Integer; *y2* : Integer)

JScript:

Function Rectangle (*x1*, *y1*, *x2*, *y2*)

Related Topics: Ellipse Procedure, LineTo Procedure, MoveTo Procedure, Draw Procedure, RoundRectangle Procedure, StretchDraw Procedure, TextHeight Function, TextOut Procedure, TextWidth Function

RefreshScreen Procedure

Applies to: TTermScreen Class.

Repaint the screen in its entirety. The **RefreshScreen** procedure should be used after one or more **SetScreenText** procedure calls in order to see all information painted on the screen.

BasicScript:

Sub RefreshScreen ()

PascalScript:

Procedure RefreshScreen

JScript:

Function RefreshScreen()

Note: Normally, **RefreshScreen** should only be used after the last **SetScreenText** call since **RefreshScreen** has to paint the entire screen and, as such, takes longer to execute than other screen handling commands.

Related Topics: SetScreenText

RemoveFolder Function

Remove an existing folder.

BasicScript:

Function RemoveFolder (ByVal *FolderName* as String) as Boolean

PascalScript:

Function RemoveFolder (*FolderName* : String) : Boolean

JScript:

Function RemoveFolder (*FolderName*)

Returns True if successful, else False.

RenameFile Function

Rename an existing file.

BasicScript:

Function CopyFile (ByVal *CurrentFileName* as String, ByVal *NewFileName* as String) as Boolean

PascalScript:

Function CopyFile (*CurrentFileName* : String, *NewFileName* : String) : Boolean

JScript:

Function CopyFile (*CurrentFileName*, *NewFileName*)

Returns True if successful, else False.

ReplaceStrings Function

Return a string after replacing specified substrings with a specfied string.

BasicScript:

Function ReplaceStrings (ByVal *s* as String, ByVal *StrToReplace* as String, ByVal *ReplaceWith* as String) as String

PascalScript:

Function ReplaceStrings (*s* : String, *StrToReplace* : String, *ReplaceWith* : String) : String

JScript:

Function ReplaceStrings(*s*, *StrToReplace*, *ReplaceWith*)

Right Function

Return a string containing the specified number of characters from the right side of a string.

BasicScript:

Function Right (ByVal *StrVal* as String, ByVal *Count* as Integer) as String

PascalScript:

Function Right (*StrVal* : String, *Count* : Integer) : String

JScript:

Function Right (*StrVal*, *Count*)

The *StrVal* parameter is the string expression from which the rightmost characters are returned.

The *Count* parameter is the numeric expression indicating the number of characters that will be returned.

Related Topics: Left Function, Len Function, Mid Function

BasicScript Example:

```
' The example uses the Right function to return the first
' of two words input by the user.

    Dim LWord, Msg, RWord, SpcPos, UsrInp               ' Declare variables
    Msg = "Enter two words separated by a space."
    UsrInp = InputBox(Msg, "Enter two words")           ' Get user input
    SpcPos = InStr(1, " ", UsrInp)                      ' Find space
    If SpcPos Then
        LWord = Left(UsrInp, SpcPos - 1)                ' Get left word
        RWord = Right(UsrInp, Len(UsrInp) - SpcPos)
                                                        ' Get right word
        Msg = "The first word you entered is <" & LWord & ">"
        Msg = Msg & RWord & "."
    Else
        Msg = "You didn't enter two words."
    End If
    MsgBox (Msg)                                        ' Display message
```

Round Function

Return rounded version on a numeric expression.

BasicScript:

Function Round (ByVal *e* as Extended) as Integer

PascalScript:

Function Round (*e* : Extended) : Integer

JScript:

Function Round (*e*)

RoundRectangle Procedure

Applies to: TCanvas Class.

Draws a rounded rectangle bounded by the specified x1, y1, x2, y2 coordinates using the current brush and pen. The x3 and y3 specify the x and y radii of the corners.

BasicScript:

Sub RoundRectangle (ByVal *x1* as Integer, ByVal *y1* as Integer, ByVal *x2* as Integer, ByVal *y2* as Integer, ByVal *x3* as Integer, ByVal *y3* as Integer)

PascalScript:

Procedure RoundRect (*x1* : Integer; *y1* : Integer; *x2* : Integer; *y2* : Integer; *x3* : Integer; *y3* : Integer)

JScript:

Procedure RoundRect (*x1*, *y1*, *x2*, *y2*, *x3*, *y3*)

Related Topics: Ellipse Procedure, LineTo Procedure, MoveTo Procedure, Rectangle Procedure, Draw Procedure, StretchDraw Procedure, TextHeight Function, TextOut Procedure, TextWidth Function

RTrim Function

Returns a string form a string trimmed of spaces from the right side.

BasicScript:

Function RTrim (ByVal *s* as String) as String

PascalScript:

Function RTrim (*s* : String) : String

JScript:

Function RTrim (*s*)

Related Topics: Trim Function,

Examples: See Trim Function

SaveScreen Procedure

Applies to: TTermScreen Class.

Save an entire screen/form to a file.

BasicScript:

Sub SaveScreen (ByVal *FileName* as String)

PascalScript:

Procedure SaveScreen (*FileName* : String)

JScript:

Function SaveScreen (*FileName*)

The *FileName* parameter is any string expression containing the file name to receive the screen/form. This capability is used commonly to save forms to files in the T27 environment and reload them (specific form loads can be assigned to function keys) from files rather than from the host. The file format is binary and only usable by the emulator.

Related topic: LoadScreen Procedure

Example:

```
#Language BasicScript
Explicit
Dim dlg
  dlg = New TOpenDialog(Self)
  Try
    If dlg.execute Then
      If Not TermScreen.SaveScreen(dlg.FileName) Then
        MsgBox("Not saved")
      Else
        MsgBox("Saved")
      End If
        End If
  Finally
    Dlg.Free
  End Try
```

ScreenAvailable Function

Applies to: TTermScreen Class.

Determine if a screen is available. Returns a 1 (true) is the specified screen name is available (configured with a route).

BasicScript:

Function ScreenAvailable (ByVal *ScreenName* as String) as Boolean

PascalScript:

Function ScreenAvailable (*ScreenName* : String) : Boolean

JScript:

Function ScreenAvailable (*ScreenName*)

The *ScreenName* parameter is an string expression. If an invalid screen name is entered, it is ignored.

Related Topics: ActivateScreen, ScreenOpen

Example: See ActivateScreen.

ScreenOpen Function

Applies to: TTermScreen Class.

Open a screen. Returns a 1 (true) if the specified screen number is currently open.

BasicScript:

Function ScreenOpen (ByVal *ScreenName* as String) as Boolean

PascalScript:

Function ScreenOpen (*ScreenName* : String) : Boolean

JScript:

Function ScreenOpen (*ScreenName*)

The *ScreenName* parameter is any string expression. If an invalid screen name is entered, it is ignored.

Related Topics: ActivateScreen, ScreenAvailable

Example: See ActivateScreen.

Send Procedure

Applies to: TTermScreen Class.

Send (Xmits) text to the host without putting the text on the screen.

BasicScript:

Sub Send (ByVal *TextToSend* as String) as String

PascalScript:

Procedure Send (*TextToSend* : String)

JScript:

Function Send (*TextToSend*)

SendKeys Procedure

Send one or more keystrokes to the titled window as if they had been entered at the keyboard.

BasicScript:

Sub SendKeys (ByVal *Keys* as String, ByVal *WindowTitle* as String = "", ByVal *Delay* as Integer = 0) as String

PascalScript:

Procedure SendKeys(*Keys* : String, *WindowTitle* : String = "", *Delay* : integer = 0)

JScript:

Function SendKeys(*Keys*, *WindowTitle* as String = "", *Delay* as Int = 0)

The *Keys* parameter is a string and is sent to the active window.

To send a single keyboard character, use the character itself. To send the letter A, use "A". To send multiple keyboard characters, one behind the other, include them in the string in the order you want them sent. To send a D followed by an E and then followed by an F, use "DEF".

Ten keyboard characters have special significance when used with the **SendKeys** statement:

Characters	Usage
{}	Braces are used to enclose a special character or key name being sent. For example, {F4} sends function key 4.
+	The plus sign is the SHIFT key.
^	The caret is the CTRL key.

%	The percent sign is the ALT key.
~	The tilde is the ENTER key.
()	Parentheses are used to enclose multiple keystrokes in combination with the SHIFT, CTRL and ALT keys. For example, "%(EF)" would be the same as holding down the ALT key while pressing E followed by F.
[ ]	No special significance but must be enclosed in braces when sent; e.g., "{[} and {]}".

To send any special character, enclose it in braces. For example, "{{}" sends an open brace and "{+}" sends a plus sign.

To send keys that do not display when you press them, use the following substitution codes:

Key	Substitution Code
BACKSPACE	{BACKSPACE}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
:	:
F16	{F16}

To repeat a key, follow the key by the number of times to repeat the keystroke. For example, "{UP 10}" is the same as pressing the UP ARROW 10 times. Note: A space is required between the key and the number.

Example:

```
X = Shell("Calc.exe", "", "", 1)               ' Shell Calculator.
' Wait for Calculator to get comletely started
CalcStarted = False
For x = 1 To 10 ' 10 Tries
    If AppActivate("Calculator") Then
        CalcStarted = true
        Break
    End If
    Wait(500)                                  ' Half second wait
Next

If Not CalcStarted Then
    MsgBox("Calculator did not start.")
    Exit
End If

' Send keystrokes to Calculator.
SendKeys ("12345", "Calculator")
SendKeys ("(+)", "Calculator")
SendKeys ("5", "Calculator")
```

```
SendKeys ("=", "Calculator")

MsgBox ("Choose OK to close the Calculator.")  ' Display OK prompt.

SendKeys ("%{F4}", "Calculator")               ' Alt+F4 to close Calculator.
```

SendMail Procedure

Display an e-mail dialog.

BasicScript:

Sub SendMail (ByVal *Recipients* as String, ByVal *Subject* as String, ByVal *CcRecipients* as String, ByVal *BccRecipients* as String, ByVal *MessageText* as String, ByVal *Attachments* as String, ByVal *NoPrompt* as Boolean)

PascalScript:

Procedure SendMail (*Recipients* : String, *Subject* : String, *CcRecipients* : String, *BccRecipients* : String, *MessageText* : String, *Attachments* : String, *NoPrompt* : Boolean)

JScript:

Function SendMail (*Recipients*, *Subject*, *CcRecipients*, *BccRecipients*, *MessageText*, *Attachments*, *NoPrompt*)

SetLength Procedure

Set the length of a specified string.

BasicScript:

Sub SetLength (ByRef *S* as String, ByVal *L* as Integer)

PascalScript:

Procedure SetLength (var *S* : String; *L* : Integer)

JScript:

Function SetLength (*S*, *L*)

SetCursor Procedure

Applies to: TTermScreen Class.

Set the column and row position of the text cursor within the logical screen.

BasicScript:

Sub SetCursor (ByVal *Col* as Integer, ByVal *Row* as Integer)

PascalScript:

Function SetCursor (*Col* : Integer, *Row* : Integer)

JScript:

SetCursor (C*ol*, *Row*)

SetScreenText Procedure

Applies to: TTermScreen Class.

Set the string value of an area within the logical screen. This procedure will set text regardless of protected FCCs in the screen.

BasicScript:

Sub SetScreenText (ByVal *Col* as Integer, ByVal *Row* as Integer = 0, ByVal *Len* as Integer, ByVal *Value* as String)

Pascal

Procedure SetScreenText (*Col* : Integer, *Row* : Integer, *Len* : Integer = 0, *Value* : String)

Jscript:

Function SetScreenText (C*ol*, *Row*, *Len* as Int = 0, *Value*)

The *Col*, *Row* and *Len* parameters are any integer expression. The *Value* parameter is any string expression.

If *Row* is specified as -1, then *Col* is assumed to be the offset from the beginning of the logical screen buffer. For example:

```
SetScreenText 5, 2, 5, "text"
```

Is the same as:

```
SetScreenText 85, -1, 5, "text"
```

The above example assumes the screen has 80 columns.

Related Topics: GetScreenText Function, GetScreenLine Function, RefreshScreenprocedure

Example:

```
' Put the trans code in the screen
SetScreenText 1, 1, 6, "CUST1 "
```

Example 2:

The SetScreenText can be used to issue UTS control sequences (Unisys ClearPath 2200 Servers only) in a script at the beginning of the screen allowing control page updates or setting FCCs in the screen.

Control characters are always entered as symbolic names enclosed within angle brackets *<>". If a "<" character is needed, it can be entered as

The following Control Sequence moves the cursor to the home position, clears the entire screen and then sets up an FCC field with video off to allow hidden entry of a user id and password:

<ESC>e<ESC>m<US> E@

The user id and password would be entered with another SetScreenText.

The following sequence would restore video on:

<ESC>e<ESC>m<US> D@

SetSessionVar Procedure

Applies to: TTermScreen Class.

Set the contents of a global session variable.

BasicScript:

Sub SetSessionVar (ByVal *VarName* as String, ByVal *VarValue* as Variant)

PascalScript:

Procedure SetSessionVar (*VarName* : String, *VarValue* : Variant)

JScript:

Function SetSessionVar (*VarName*, *VarValue*)

The *VarName* parameter is any string expression containing the session variable. The *VarValue* parameter is the string expression to be assigned to the named session variable.

Related Topics: GetSessionVar Function

Example: See GetSessionVar Function

SetVariable Procedure

Applies to: TDialogForm Class.

This method allows the script to initialize the value of a variable defined in the Dialog Form's action script. The specified variable must be declared Global in the Dialog Form's action script.

BasicScript:

Sub SetVariable (ByVal *VarName* as String, ByVal *VarValue*)

PascalScript:

Procedure SetVariable (*VarName* : String, *VarValue* : Variant)

JScript:

Function SetVariable (*VarName*, *VarValue*)

Related Topics: Free Procedure, LoadForm Function, Create Function, GetVariable Function, ShowForm Function, ClearForm Procedure, PrintForm Procedure

Shell Procedure

12345Start another application or open a file with its associated application.

BasicScript:

Sub Shell (ByVal *ProgramFile* as String, ByVal *Parameters* as String = "", ByVal *StartInDir* as String = "", ByVal *Style* as Integer = 1)

PascalScript:

Procedure Shell (*ProgramFile* : String, *Parameters* : String = "", *StartInDir* : String = "", *Style* : Integer = 1)

JScript:

Function Shell (*ProgramFile*, *Parameters* as String = "", *StartInDir* as String = "", *Style* as Int = 1)

The Shell function has four parameters. The first one, *ProgramFile*, is the name of the program to be executed. Note: The extension name (.BAT, .EXE, etc.) must be included or an error will occur. The second parameter is the list of any parameters to be passed to the program. The third parameter is path of the directory in which the program will start. The fourth argument, *Style*, is the number corresponding to the style of the window. The fourth argument is also optional, and if omitted, the program is opened in a normal window with focus.

Value	Window Style
1, 5, 9	Normal with focus.
2	Minimized with focus (default).
3	Maximized with focus.
4, 8	Normal without focus.
6, 7	Minimized without focus.

BasicScript Example:

```
X = Shell("Calc.exe", "", "", 1)                ' Shell Calculator.
' Wait for Calculator to get comletely started
CalcStarted = False
For x = 1 To 10 ' 10 Tries
    If AppActivate("Calculator") Then
        CalcStarted = true
        Break
    End If
    Wait(500)                                   ' Half second wait
Next

If Not CalcStarted Then
    MsgBox("Calculator did not start.")
    Exit
End If

' Send keystrokes to Calculator.
SendKeys ("12345", "Calculator")
SendKeys ("(+)", "Calculator")
SendKeys ("5", "Calculator")
SendKeys ("=", "Calculator")

MsgBox ("Choose OK to close the Calculator.")  ' Display OK prompt.

SendKeys ("%{F4}", "Calculator")               ' Alt+F4 to close Calculator.
```

ShowForm Function

Applies to: TDialogForm Class.

This method causes the Dialog Form to be shown modally. Modal means that the current script will wait for the Dialog Form to be closed before continuing to execute. The result will be whatever is set by the Dialog Form.

BasicScript:

Function ShowForm () as Integer

PascalScript:

Function ShowForm() : Integer

JScript:

Function ShowForm()

When the DialogForm.ShowForm is method is called, its result type is a TModalResult. TModalResult represents the value returned by a modal dialog — in this case the Dialog Form. An application can

use any integer value as a modal result value. Although TModalResult can take any integer value, the following constants are defined for commonly used TModalResult values:

Constant	Integer
mrNone	0
mrOK	1
mrCancel	2
mrAbort	3
mrRetry	4
mrIgnore	5
mrYes	6
mrNo	7
mrAll	8
mrNoToAll	9
mrYesToAll	10

Setting ModalResult to anything other than 0 causes the Dialog to close and return the specified value.

Modal means that the calling script waits until the Dialog Form is closed at the point of calling the ShowFrom. If the Dialog were shown non-modally, the script would get an immediate return while the dialog is still active. The ShowFormNonModal method can be used to show the form in a non-modal fashion. If shown non-modal, the script would have to loop until the form is closed, because as soon as the script ends the Dialog would close.

See also, ModalResult Clarification and More.

Related Topics: Free Procedure, LoadForm Function, SetVariable Procedure, GetVariable Function, Create Function, ClearForm Procedure, PrintForm Procedure, ShowFormNonModal Procedure, TDialogForm Class

Example: See Create Function.

ShowFormNonModal Procedure

Applies to: TDialogForm Class.

This method shows a Dialog Form in a non-modal state, meaning the script does not stop and wait for a Dialog Form to close. To use a non-modal dialog, the script has to keep itself alive, using loops or something, until time to close the form.

BasicScript:

Sub ShowFormNonModal ()

PascalScript:

Procedure ShowFormNonModal

JScript:

Function ShowFormNonModal

Related Topics: Free Procedure, LoadForm Function, SetVariable Procedure, GetVariable Function, Create Function, ClearForm Procedure, PrintForm Procedure, ShowForm Function, TDialogForm Class

BasicScript Example:

```
' This example demonstrats how a Non Modal dialog may be used.
' In this case the script shows a sign on dialog but continues to process
' while the used enters a user-id and password.  A seriers of Loops
' containing wait statements prevents the script from getting ahead of
' the user's entry.

  Df = New TDialogForm(Self)
  Df.LoadForm("SignOnInProcess")
  Df.ShowFormNonModal              '  ←This will show the dialog,
                                   '    but the script will continue
  Try
    GotEnterUserId = False
    GotPrev = Flase

    ' Loop here until the "Enter your password" propt is on the screen.
    ' This loop is limited to 15 iterations or 15 seconds.
    ' This can take place while the user is typing into the dialog.
    For x = 1 To 15
```

```
      Df.Pnl_Prog.Caption = Df.Pnl_Prog.Caption + "*" ' Show some activity
      If TermScreen.GetScreenText(2, 23, 18) = "Enter your user-id" Then
        GotEnterUserId = true
        Break
      End If
      Wait(1000)
    Next

    If GotEnterUserId Then  ' We can't continue because
      ' the "Enter your user-id" prompt was not received.
      ' Loop here until user has entered a user-id and password and clicks OK.
      ' This loop is not limited.
      While true
        ' This Wait is important--if no Wait is done the terminal emulator
        ' will not be able to process incoming messages and appear to be hung.
        Wait(100)
        If Df.Btn_OK.Tag = 1 Then
          Break
        End If
      Wend

      ' Send the useds name and password
      TermScreen.Send(Trim(Df.Ed_UserId.Text) + "/" + Trim(Df.Ed_Password.Text))

      ' Loop here until the "Previos session" message is on the screnn.
      ' This loop is limited to 10 iterations or 10 seconds.
      For x = 1 To 10
        Df.Pnl_Prog.Caption = Df.Pnl_Prog.Caption + "*"  ' Show some activity
        If TermScreen.GetScreenText(2, 23, 16) = "Previous session" Then
          GotPrev = True
          Break
        End If
        Wait(1000)
      Next
    End If

  Finally
    Delete Df
  End Try

  If GotPrev Then
    MsgBox("You are now signed on the the mainframe.  Have fun.", mb_IconInformation,
"Sign On Complete")
  Else
    MsgBox("A promblem has occurred while trying to sign on the the
mainframe.  Contact technical support", mb_IconExclamation, "Sign On Failed")
  End If
```

Note: See the SignOnInProcess.BFM and .ACT in the sample dialogs of the installed scripts directory.

ShowMessage Procedure

Show a modal message box.

BasicScript:

Sub ShowMessage (ByVal *Msg* as Variant)

PascalScript:

Procedure ShowMessage (*Msg* : Variant)

JScript:

Function ShowMessage (*Msg*)

Sin Function

Return the sin of an angle.

BasicScript:

Function Sin (ByVal *e* as Extended) as Extended

PascalScript:

Function Sin (*e* : Extended) : Extended

JScript:

Function Sin (*e*)

BasicScript Example:

```
rad = 90 * (Pi()/180)  ' Rad
x = Sin(rad)         ' Sin
MsgBox (x)
```

Space Function

Returns a string of a specified length containing all spaces (MakeString).

BasicScript:

Function Space (ByVal *Length* as Integer) as String

PascalScript:

Function Space (*Length* : Integer) : String

JScript:

Function Space (*Length*)

The *Length* parameter can be any valid integer and determines the number of blanks.

BasicScript Example:

```
MsgBox ("Hello" & Space(20) & "There")
```

Sqrt Function

Return the square root of a numeric expression.

BasicScript:

Function Sqrt (ByVal *e* as Extended) as Extended

PascalScript:

Function Sqrt (*e* : Extended) : Extended

JScript:

Function Sqrt (*e*)

The *e* parameter must be a valid number greater than or equal to zero.

BasicScript Example:

```
Dim Msg, Number                                  ' Declare variables.
Msg = "Enter a non-negative number."
Number = InputBox("Square Root Calc",Msg)        ' Get user input.
If Number < 0 Then
    Msg = "Cannot determine the square root of a negative number."
Else
    Msg = "The square root of " & Number & " is "
    Msg = Msg & Sqrt(Number) & "."
End If
MsgBox (Msg)                                     ' Display results.
```

Str Function

Return the value of a numeric expression.

BasicScript:

Function Str (ByVal n as Variant) as Variant

PascalScript:

Function Str (n : Variant) : Variant

JScript:

Function Str (n)

Str returns a String.

Use the Format, FormatDate, FormatFloat and FormatMaskText functions to convert numeric values you want formatted as dates, times or in other user-defined formats.

Related topics: Format Function, Val Function

BasicScript Example:

```
Dim msg
a = -1
msgBox ("Num = " & Str(a))
MsgBox ("_Abs(Num) =" & Str(_Abs(a)))
```

StretchDraw Procedure

Applies to: Tcanvas Class.

Draw a graphic within the specified x1, y1, x2, y2 coordinates. The graphic's dimensions will be stretched/shrunk to fit the specified rectangle.

BasicScript:

Sub StretchDraw(ByVal *x1* as Integerv ByVal *y1* as Integer, ByVal *x2* as Integer, ByVal *y2* as Integer, ByVal *Graphic* as Tgraphic)

PascalScript:

Procedure StretchDraw(*x1* : Integer; *y1* : Integer; *x2* : Integer; *y2* : Integer, *Graphic* : Tgraphic)

JScript:

Function StretchDraw(*x1*, *y1*, *x2*, *y2*, *Graphic*)

Related Topics: Ellipse Procedure, LineTo Procedure, MoveTo Procedure, Rectangle Procedure, RoundRectangle Procedure, Draw Procedure, TextHeight Function, TextOut Procedure, TextWidth Function

StrToDate Function

Convert a string to a date.

BasicScript:

Function StrToDate (ByVal *s* as String) as Extended

PascalScript:

Function StrToDate (*s* : String) : Extended

JScript:

Function StrToDate (*s*)

StrToDateTime Function

Convert a string to date and time.

BasicScript:

Function StrToDateTime (ByVal *s* as String) as Extended

PascalScript:

Function StrToDateTime (*s* : String) : Extended

JScript:

Function StrToDateTime (*s*)

StrToFloat Function

Convert a string to a floating-point value.

BasicScript:

Function StrToFloat (ByVal *s* as String) as Extended

PascalScript:

Function StrToFloat (*s* : String) : Extended

JScript:

Function StrToFloat (*s*)

StrToInt Function

Convert a string to an integer value.

BasicScript:

Function StrToInt (By Val *s* as String) as Integer

PascalScript:

Function StrToInt (*s* : String) : Integer

JScript:

Function StrToInt (*s*)

StrToTime Function

Convert a string to time.

BasicScript:

Function StrToTime (ByVal s as String) as Extended

PascalScript:

Function StrToTime (s : String) : Extended

JScript:

Function StrToTime (s)

SwitchToolBar Procedure

Switch a configured toolbar.

BasicScript:

Sub SwitchToolBar (ByVal *ToolBarName* as String, ByVal *ToolBarNumber* as Integer = 1, ByVal *ShowIt* as Boolean)

PascalScript:

Procedure SwitchToolBar (*ToolBarName* : String, *ToolBarNumber* : Integer = 1, *ShowIt* : Boolean = True)

JScript:

Function SwitchToolBar (*ToolBarName*, *ToolBarNumber as Int = 1*, *ShowIt*)

If *ToolBarName* is set to an asterisk (*), the Select Toolbar dialog will be shown. The Select Toolbar dialog only affects toolbar line 1, so the second parameter is not used (see example, below).

ToolBarNumber must be between 1 and 3. If it is not, it will be set to 1 automatically.

Note: The last selected toolbar(s) will be saved when the screen is closed and restored when the screen is reopened.

Examples:

```
TermScreen.SwitchToolBar("MyToolBar")
    ' Switches toolbar 1 to MyToolBar and shows the toolbars
TermScreen.SwitchToolBar("MyToolBar", 2, True)
    ' Switches the second tool bar line and shows the toolbars
TermScreen.SwitchToolBar("default",1,False)
    ' Switches the first tool bar to DEFAULT and hides the toolbar.
TermScreen.SwitchToolBar("",1,False)
    ' Clears the first toolbar and hides the toolbar.
TermScreen.SwitchToolBar("*",,true)
    ' Shows the Select Toolbar dialog.
```

Tan Function

Return the tangent of an angle.

BasicScript:

Function Tan (ByVal *X* as Extended) as Extended

PascalScript:

Function Tan (*X* : Extended) : Extended

JScript:

Function Tan (*X*)

The *X* parameter must be a valid angle expressed in radians.

Related Topic: ArcTan Function, Cos Function, Sin Function

BasicScript Example:

```
Dim Msg                  ' Declare variable
Msg = "Pi is equal to " & FloatToStr(Pi())
MsgBox (Msg)
x = Tan(Pi())/4)
y = FloatToStr(x) & " is the tangent of Pi/4"
MsgBox (y)                ' Display results
```

TextHeight Function

Applies to: TCanvas Class.

Return the pixel height of the specified text using the current printer and font settings.

BasicScript:

Function TextHeight (ByVal *Text* as String) as Integer

PascalScript:

Function TextHeight (*Text* : String) : Integer

JScript:

Function TextHeight (*Text*)

Related Topics: Ellipse Procedure, LineTo Procedure, MoveTo Procedure, Rectangle Procedure, RoundRectangle Procedure, StretchDraw Procedure, Draw Procedure, TextOut Procedure, TextWidth Function

TextOut Procedure

Applies to: TXSPrint Class and TCanvas Class.

TXSPrint Class:

Write the specified text to the printer page at the specified x and y pixel coordinates. The current drawing x and y positions are not changed.

TCanvas Class:

Writes the specified Text string are the specified x, y coordinates.

BasicScript:

Sub TextOut (ByVal *x* as Integer, ByVal *y* as Integer, ByVal *Text* as String)

PascalScript:

Procedure TextOut (*x* : integer; *y* : integer; *Text* : String)

JScript:

Function TextOut (*x, y, Text*)

Related Topics: EndDoc Procedure, BeginDoc Procedure, PrintLine Procedure, LineSpace Procedure, Abort Procedure, GetTextHeight Function, GetTextWidth Function, NewPage Procedure, MoveTo Procedure, LineTo Procedure, DrawRect Procedure

TextWidth Function

Applies to: TCanvas Class.

Return the pixel width of the specified text using the current printer and font settings.

BasicScript:

Function TextWidth (ByVal *Text* as String) as Integer

PascalScript:

Function TextWidth (*Text* : String) : Integer

JScript:

Function TextWidth (*Text*)

Related Topics: Ellipse Procedure, LineTo Procedure, MoveTo Procedure, Rectangle Procedure, RoundRectangle Procedure, StretchDraw Procedure, TextHeight Function, TextOut Procedure, Draw Procedure

Time Function

Return the current system time.

BasicScript:

Function Time () as TDateTime

PascalScript:

Function Time() : TDateTime

JScript:

Function Time()

BasicScript Example:

```
x = Time()
MsgBox (x)                              ' Long time hh:mm:ss
MyStr = FormatDateTime("t", Time())     ' Short time hh:mm
MsgBox (MyStr)
MyStr = FormatDateTime("tt", Time())    ' Long time hh:mm:ss
MsgBox (MyStr)
```

TimeToStr Function

Convert time to string.

BasicScript:

Function TimeToStr (ByVal *e* as Extended) as String

PascalScript:

Function TimeToStr (*e* : Extended) : String

JScript:

Function TimeToStr (*e*)

Trim Function

Return a copy of a string with leading, trailing or both leading and training spaces removed.

BasicScript:

Function Trim (ByVal *s* as String) as String

PascalScript:

Function Trim (*s* : String) : String

JScript:

Function Trim (*s*)

Trim removes leading and trailing spaces.

Related Topics: LTrim Function,

BasicScript Example:

```
' This example uses the LTrim and RTrim functions to strip leading
' and trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the
' use of nested function calls.

    MyString = "  <-Trim->  "             ' Initialize string
    TrimString = LTrim(MyString)          ' TrimString = "<-Trim->  "
    MsgBox ("|" & TrimString & "|")
    TrimString = LCase(RTrim(MyString))   ' TrimString = "  <-trim->"
    MsgBox ("|" & TrimString & "|")
    TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->"
    MsgBox ("|" & TrimString & "|")       ' Using the Trim function
                                          ' alone achieves the same
                                          ' result.
    TrimString = UCase(Trim(MyString))    ' TrimString = "<-TRIM->"
    MsgBox ("|" & TrimString & "|")
```

Trunc Function

Return the truncated value of a numeric expression.

BasicScript:

Function TimeToStr (ByVal *e* as Extended) as String

PascalScript:

Function TimeToStr (*e* : Extended) : String

JScript:

Function TimeToStr (*e*)

UCase Function

Returns the value of a string converted to upper case (UpperCase).

BasicScript:

Function UCase (ByVal *s* as String) as String

PascalScript:

Function UCase (*s* : String) : String

JScript:

Function UCase (*s*)

Related Topics: LCase Function, Uppercase Function, Lowercase Function

Example:

```
' This example uses the LTrim and RTrim functions to strip leading
' and trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the
' use of nested function calls

    MyString = "  <-Trim->  "             ' Initialize string
    TrimString = LTrim(MyString)          ' TrimString = "<-Trim->  "
    MsgBox ("|" & TrimString & "|")
    TrimString = LCase(RTrim(MyString))   ' TrimString = "  <-trim->"
    MsgBox ("|" & TrimString & "|")
    TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->"
    MsgBox ("|" & TrimString & "|")       ' Using the Trim function
                                          ' alone achieves the same
                                          ' result.
    TrimString = UCase(Trim(MyString))    ' TrimString = "<-TRIM->"
    MsgBox ("|" & TrimString & "|")
```

Uppercase Function

Returns the value of a string converted to upper case (UCase).

BasicScript:

Function Uppercase (ByVal *s* as String) as String

PascalScript:

Function Uppercase (*s* : String) : String

JScript:

Function Uppercase (*s*)

Related Topics: UCase Function, Lowercase Function, LCase Function

BasicScript Example:

```
' This example uses the LTrim and RTrim functions to strip leading
' and trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the
' use of nested function calls

    MyString = "  <-Trim->  "             ' Initialize string
    TrimString = LTrim(MyString)          ' TrimString = "<-Trim->  "
    MsgBox ("|" & TrimString & "|")
    TrimString = Lowercase(RTrim(MyString)) ' TrimString = "  <-trim->"
    MsgBox ("|" & TrimString & "|")
    TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->"
    MsgBox ("|" & TrimString & "|")       ' Using the Trim function
                                          ' alone achieves the same
                                          ' result.
    TrimString = Uppercase(Trim(MyString)) ' TrimString = "<-TRIM->"
    MsgBox ("|" & TrimString & "|")
```

Val Function

Return the numeric value of a variant.

BasicScript:

Function Val (ByVal *v* as Variant) as Variant

PascalScript:

Function Val (*v* : Variant) : Variant

JScript:

Function Val (*v*)

BasicScript Example:

```
Dim Msg, A
Dim YourVal As Double
YourVal = Val(InputBox("Enter a number","",""))
A = YourVal
Msg = "The number you enered is: " & A
MsgBox (Msg)
```

ValidDate Function

Validate a date value.

BasicScript:

Function ValidDate (ByVal *cDate* as String) as Boolean

PascalScript:

Function ValidDate (*cDate* : String) : Boolean

JScript:

Function ValidDate (*cDate*)

ValidFloat Function

Validate a floating-point value.

BasicScript:

Function ValidFloat (ByVal *cFlt* as String) as Boolean

PascalScript:

Function ValidFloat (*cFlt* : String) : Boolean

JScript:

Function ValidFloat (*cFlt*)

ValidInt Function

Validate an integer value.

BasicScript:

Function ValidInt (ByVal *cInt* as String) as Boolean

PascalScript:

Function ValidInt (*cInt* : String) : Boolean

JScript:

Function ValidInt (*cInt*)

VarArrayCreate Function

Create a variant array.

BasicScript:

Function VarArrayCreate (ByVal *Bounds* as Array, ByVal *Typ* as Integer) as Variant

PascalScript:

Function VarArrayCreate (*Bounds* : Array; *Typ* : Integer) : Variant

JScript:

Function VarArrayCreate (*Bounds*, *Typ*)

VarToStr Function

Convert a variant to a string.

BasicScript:

Function VarToStr (ByVal *v* as Variant) as String

PascalScript:

Function VarToStr (*v* : Variant) : String

JScript:

Function VarToStr (*v*)

VarTypeToStr Function

Return the variant type name of a specified variant.

BasicScript:

Function VarTypeToStr (ByVal *VarType* as Variant) as String

PascalScript:

Function VarTypeToStr (*VarType* : Variant) : String

JScript:

Function VarTypeToStr (*VarType*)

Wait Procedure

Cause a timed wait. Wait for a fixed amount of time. A call to this procedure causes the script to wait for a period before continuing on to the next script statement, procedure or function.

BasicScript:

Sub Wait (ByVal *milliseconds* as Integer)

PascalScript:

Procedure Wait (*milliseconds* : Integer)

JScript:

Function Wait (*milliseconds*)

The *milliseconds* parameter is an integer expression containing the amount of time to wait expressed in milliseconds.

Example:

(see GetScreenLine Function).

WaitForSpecificString Function

Applies to: TTermScreen Class.

Cause the script to wait for the specified *string* at the specified location on the screen.

BasicScript:

Function WaitForSpecificString (ByVal *row* as Integer, ByVal *col* as Integer, ByVal *len* as Integer, ByVal *string* as String) as Integer

PascalScript:

Function WaitForSpecificString (*row* : Integer, *col* : Integer, *len* : Integer, *string* : String) : Integer

JScript:

Function WaitForSpecificString(*row*, *col*, *len*, *string*)

The *col*, *row* and *len* parameters are any integer expression. The *string* parameter is any string expression.

If the *len* parameter is zero (0), the length of the specified *string* will be used.

Related Topics: GetLastMsg , GetScreenLine, GetScreenText, WaitForString

Example:

(see GetScreenText).

WaitForString Function

Applies to: TTermScreen Class.

Cause the script to wait for the specified *string*. Like the **GetLastMsg** function, **WaitForString**, only retrieves the first 80 characters of the last message received (including any control sequences) from the host or communication system.

BasicScript:

Function WaitForString (ByVal *string* as String) as Integer

PascalScript:

Function WaitForString (*string* : String) : Integer

JScript:

Function WaitForString (*string*)

The communication system may return multiple messages before control is returned to the script; therefore, only the last message is accessible by this function.

Since the message may contain control sequences, this function may not be very useful unless you are familiar with the handling of control sequences by the communications system. Consider using the WaitForSpecificString function.

Related Topics: GetLastMsg , GetScreenLine, GetScreenText, WaitForSpecificString

WaitString Function

Applies to: TTermScreen Class.

Cause the script to wait for the specified Target at the specified location with NotEqual and TimeOut values.

BasicScript:

Function WaitString (ByVal *Target* as String, ByVal *Col* as Integer, ByVal *Row* as Integer, ByVal *NotEqual* as Integer = 0, ByVal *TimeOut* as Integer = 5) as Integer

PascalScript:

Function WaitString (*Target* : String, *Col* : Integer, *Row* : Integer, *NotEqual* : Integer = 0, *TimeOut* : Integer = 5) : Integer

JScript:

Function WaitString (*Target*, *Col*, *Row*, *NotEqual* as Int = 0, *TimeOut* as Int = 5)

The following is the list of parameters and their purpose:

Parameter	Purpose
Target	The string you are waiting for.
Col	The column where it is to be search.
Row	The row where it is to be searched. If both *Row* and *Col* are 0, the entire screen is searched. If *Col* is 0, the entire row will be searched.
NotEqual	A find is made when the specified position does NOT contain the target (*Col* and *Row* must be specified). *NotEqual* = 0 means the string must match. *NotEqual* = 1 means the string must not match (i.e., you want to wait until something on the screen is overwrtten).
TimeOut	Time to wait specified in seconds.

Returns True if condition is met within the timeout period.

WriteLine Procedure

Applies to: TXSTextFile Class.

Write the specified line to the currently open file.

BasicScript:

Sub WriteLine (ByVal *ErrorStatus* as Integer, ByVal *Line* as String)

PascalScript:

Procedure WriteLine (*ErrorStatus* : Integer; *Line* : String)

JScript:

Function WriteLine (*ErrorStatus*, *Line*)

The file's FileMode on the Open function must be either fmWrite or fmAppend.

If the write is successful, ErrorStatus will contain 0,;otherwise, it will contain the system error code.

Related Topics: Open Function, Close Procedure, ReadLine Function

The following is a BasicScript example of the TSXTextFile object used to copy one text file to another:

```
dim st as integer
dim s as string
dim cnt as integer

F1 = New TXSTextFile(Self)
F2 = new TXSTextFile(Self)

try
  If Not F1.open(termscreen.scriptfolder +"\Buttons.xs", fmRead) Then
    MsgBox("File F1 open error: " + F1.LastErrorMessage)
    Exit
  End If

   F2.Open(TermScreen.ScriptFolder +"\AAAA.xx", fmWrite)

   while not F1.EOF
     s = F1.readline(st)
     if st <> 0 then
       msgbox("Error on input file: " + F1.LastErrorMessage,
mb_IconExclamation, "Input File Error")
       break
     else
       inc(cnt)
       F2.WriteLine(st, s)
       if st <> 0 then
         msgbox("Write error: " + F2.LastErrorMessage, mb_IconExclamation,
"Output File Error")
         break
       End If

     End If
   WEnd
Finally
   F1.free
   F2.free
End Try

MsgBox("Copied " + IntToStr(cnt) + " lines.", mb_IconInformation, "Copy Done")
```

Constants

Predefined Constants

This topic contains all constants and their equivalent values that are predefined when a dialog/script is invoked.

ColorDialog Options

Constant	Description
cdFullOpen	Display custom color options when the dialog opens.
cdPreventFullOpen	Disable the Define Custom Colors button in the dialog, so that the user cannot define new colors.
cdShowHelp	Add a Help button to the color dialog.
cdSolidColor	Direct Windows to use the nearest solid color to the color chosen.
cdAnyColor	Allow the user to select non-solid colors (which may have to be approximated by dithering).

File Mode Values:

Constant	Description
fmRead	Open the file for reading
fmWrite	Open the file for writing
fmAppend	Open the file for writing and append new records to the end of the file when it already exists.

GetFolderPath CLSIDs:

Constant	Integer
CSIDL_PERSONAL	5
CSIDL_APPDATA	26
CSIDL_LOCAL_APPDATA	28
CSIDL_INTERNET_CACHE	32
CSIDL_COOKIES	33
CSIDL_HISTORY	34
CSIDL_COMMON_APPDATA	35
CSIDL_WINDOWS	36
CSIDL_SYSTEM	37
CSIDL_PROGRAM_FILES	38
CSIDL_MYPICTURES	39
CSIDL_PROGRAM_FILES_COMMON	43
CSIDL_COMMON_DOCUMENTS	46

Keyboard States:

Constant	Value	Description
KEYBOARDUNLOCKED	0	Keyboard Unlocked
KEYBOARDLOCKED	1	Keyboard Locked

Message Box Constants:

Constant	Value	Description
MsgBox Buttons:		
MB_OK	0	OK button only.
MB_OKCANCEL	1	OK and Cancel buttons.
MB_ABORTRETRYIGNORE	2	Abort, Retry and Ignore buttons.
MB_YESNOCANCEL	3	Yes, No and Cancel buttons.
MB_YESNO	4	Yes and No buttons.
MB_RETRYCANCEL	5	Retry and Cancel buttons
MsgBox Icons:		
MB_ICONSTOP	16	Critical message.  For Windows 95, display:
MB_ICONQUESTION	32	Warning query.  For Windows 95, display:
MB_ICONEXCLAMATION	48	Warning message.  For Windows 95, display:
MB_ICONINFORMATION	64	Information message.

Constant	Value	Description
		 For Windows 95, display:
MsgBox Defaults:		
MB_APPLMODAL	0	Application Modal Message Box. The user must respond to the message before continuing work in the current application (Default).
MB_DEFBUTTON1	0	First button is default.
MB_DEFBUTTON2	256	Second button is default.
MB_DEFBUTTON3	512	Third button is default.
MB_SYSTEMMODAL	4096	System Modal. All applications are suspended until the user responds to the message box.
MsgBox return values:		
IDOK	1	OK button pressed.
IDCANCEL	2	Cancel button pressed.
IDABORT	3	Abort button pressed.
IDRETRY	4	Retry button pressed.
IDIGNORE	5	Ignore button pressed.
IDYES	6	Yes button pressed.
IDNO	7	No button pressed.

Message Waiting States:

Constant	Value	Description
NOMESSAGEWAITING	0	Message Not Waiting
MESSAGEWAITING	1	Message Waiting

Pen Mode Descriptions:

Mode	Pixel color
pmBlack	Always black
pmWhite	Always white
pmNop	Unchanged
pmNot	Inverse of canvas background color
pmCopy	Pen color specified in Color property
pmNotCopy	Inverse of pen color
pmMergePenNot	Combination of pen color and inverse of canvas background
pmMaskPenNot	Combination of colors common to both pen and inverse of canvas background
pmMergeNotPen	Combination of canvas background color and inverse of pen color
pmMaskNotPen	Combination of colors common to both canvas background and inverse of pen
pmMerge	Combination of pen color and canvas background color
pmNotMerge	Inverse of pmMerge: combination of pen color and canvas background color
pmMask	Combination of colors common to both pen and canvas background
pmNotMask	Inverse of pmMask: combination of colors common to both pen and canvas background
pmXor	Combination of colors in either pen or canvas background, but not both
pmNotXor	Inverse of pmXor: combination of colors in either pen or canvas background, but not both

Print Font Style Types:

Constant	Value	Description
fsNormal	0	Normal font
fsFontBold	1	Bold font
fsFontItalic	2	Italic font
fsFontUnderline	4	Underlined font
fsFontStrikeThru	8	Strikethrough font

Screen Attributes:

Constant	Value	Description
ATTR_NORMAL	0	Normal
ATTR_FIELD	1	Start of Field (set on first position of field)
ATTR_TAB	2	Tab Stop (at start of field only)

ATTR_CHANGED	4	Data Field Changed Flag
ATTR_PROTECTED	8	Protected
ATTR_VIDEO_OFF	16	Video Off
ATTR_NUMERIC	32	Numeric Only Input
ATTR_ALPHA	64	Alphabetic Only Input
ATTR_BLINK	128	Blinking
ATTR_RIGHT	256	Right Justified Data
ATTR_LOWINT	512	UTS Low Intensity
ATTR_REV	1024	Reverse Video

T27 Keys:

Constant	Integer
TK_ARROWDN	249
TK_ARROWLEFT	247
TK_ARROWRIGHT	248
TK_ARROWUP	246
TK_BACKSPACE	8
TK_BACKTAB	196
TK_BOUND	218
TK_CARRIAGERTN	13
TK_CLRALLVTAB	16442
TK_CLREOL	134
TK_CLREOP	135
TK_CLRFORMS	159
TK_CLRHOME	128
TK_COPY	16432
TK_CTRL	164
TK_CUT	16431
TK_DBLZERO	234
TK_DELCHAR	132
TK_DELCHARPAGE	16425
TK_DELLINE	133
TK_HOME	174
TK_INSCHAR	130
TK_INSCHARPAGE	16424
TK_INSLINE	131
TK_LOCAL	168
TK_LOCKCTRL	165
TK_LOGICALEOL	16415
TK_MARK	217
TK_MOVELINEDOWN	138
TK_MOVELINEUP	139
TK_NEXTPAGE	253
TK_PASTE	16434
TK_PREVPAGE	252
TK_PRINTALL	157
TK_PRINTUNPROT	156
TK_RECALL	214
TK_RECEIVE	170
TK_ROLLDN	136
TK_ROLLUP	137
TK_SETFORMS	158
TK_SPECIFY	166
TK_STORE	213
TK_TAB	198
TK_TOGGLEFORMS	141
TK_TOGGLETAB	16441
TK_TRANSMIT	172
TK_TRANSMITLINE	16428
TK_TRIPZERO	236
TK_UPPERONLYON	210
TK_UPPERONLYOFF	211
TK_WRITEESC	16426
TK_WRITEETX	3
TK_WRITEGS	16427

UTS Keys:

Constant	Value

Constant	Value
UK_BACK_SPACE	95
UK_CURSOR_DOWN	6
UK_CURSOR_LEFT	7
UK_CURSOR_RETURN_KEY	32
UK_CURSOR_RIGHT	8
UK_CURSOR_TO_END_LINE	66
UK_CURSOR_TO_HOME	23
UK_CURSOR_TO_START_LINE	65
UK_CURSOR_UP	9
UK_DELETE_IN_DISPLAY	11
UK_DELETE_IN_LINE	12
UK_DELETE_LINE	10
UK_ERASE_CHAR	67
UK_ERASE_DISPLAY	14
UK_ERASE_TO_END_DISPLAY	15
UK_ERASE_TO_END_FIELD	16
UK_ERASE_TO_END_LINE	17
UK_FKEY_1	43
UK_FKEY_2	44
UK_FKEY_3	45
UK_FKEY_4	46
UK_FKEY_5	47
UK_FKEY_6	48
UK_FKEY_7	49
UK_FKEY_8	50
UK_FKEY_9	51
UK_FKEY_10	52
UK_FKEY_11	53
UK_FKEY_12	54
UK_FKEY_13	55
UK_FKEY_14	56
UK_FKEY_15	57
UK_FKEY_16	58
UK_FKEY_17	59
UK_FKEY_18	60
UK_FKEY_19	61
UK_FKEY_20	62
UK_FKEY_21	63
UK_FKEY_22	64
UK_INSERT_IN_DISPLAY	25
UK_INSERT_IN_LINE	26
UK_INSERT_LINE	24
UK_KEYBOARD_UNLOCK	27
UK_LINE_DUP	28
UK_MSG_WAIT	29
UK_PRINT_KEY	30
UK_PRINT_ENTIRE_SCREEN	69
UK_SOE	3
UK_TAB_BACK	33
UK_TAB_FORWARD	34
UK_TAB_SET	35
UK_TRANSMIT_KEY	36

Window States:

Constant	Value	Description
WSNORMAL	0	Window Normal
WSMINIMIZED	1	Window Minimized
WSMAXIMIZED	2	Window Maximized

Index